Universidade Federal da Paraíba
Centro de Informática
Departamento de Computação Científica
Programa de Pós-Graduação em Modelagem Matemática e Computacional

Rômulo da Silva Lima

O MÉTODO DAS SOLUÇÕES FUNDAMENTAIS APLICADO À RECONSTRUÇÃO DE FONTES CONCENTRADAS PARA PROBLEMAS ELÍPTICOS

João Pessoa

2019

Catalogação na publicação
Seção de Catalogação e Classificação

L732m Lima, Rômulo da Silva.
O Método das Soluções Fundamentais aplicado à Reconstrução de Fontes Concentradas para Problemas Elípticos / Rômulo da Silva Lima. - João Pessoa, 2019.
138 f.

Orientação: Jairo Faria.
Coorientação: Thiago Machado.
Dissertação (Mestrado) - UFPB/CI/PPGMMC.

1. Método das Soluções Fundamentais. 2. Problemas Inversos de Fonte. 3. Equação de Helmholtz. 4. Equação de Poisson. I. Faria, Jairo. II. Machado, Thiago. III. Título.

UFPB/BC

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Modelagem Matemática e Computacional

O MÉTODO DAS SOLUÇÕES FUNDAMENTAIS APLICADO À RECONSTRUÇÃO DE FONTES CONCENTRADAS PARA PROBLEMAS ELÍPTICOS

Rômulo da Silva Lima

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, UFPB, da Universidade Federal da Paraíba, como parte dos requisitos necessários à obtenção do título de Mestre em Modelagem Matemática e Computacional.

Orientadores: Jairo Rocha de Faria
Thiago José Machado

João Pessoa
Março de 2019

O MÉTODO DAS SOLUÇÕES FUNDAMENTAIS APLICADO À RECONSTRUÇÃO DE FONTES CONCENTRADAS PARA PROBLEMAS ELÍPTICOS

Rômulo da Silva Lima

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL (PPGMMC) DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM MODELAGEM MATEMÁTICA E COMPUTACIONAL.

Examinada por:

Prof. Jairo Rocha de Faria, D.Sc.

Prof. Fágner Dias Araruna, D.Sc.

Prof. Hugo Leonardo Davi de Souza Cavalcante, D.Sc.

Prof.ª Sônia Maria Gomes, D.Sc.

Prof. Thiago José Machado, D.Sc.

JOÃO PESSOA, PB – BRASIL
MARÇO DE 2019

Agradecimentos

Agradeço primeiramente a Deus por ter guiado e iluminando o meu caminho para seguir em frente com meus objetivos, dando-me tranquilidade nos momentos de dificuldades. A Ele também agradeço por manter com saúde a minha família que esteve sempre ao meu lado.

À minha família pelo apoio e incentivo que me deram, especialmente à minha mãe por confiar e acreditar na minha capacidade. Sem ela, nada disso seria possível.

Um agradecimento especial à minha amada noiva Mayara Layra, por todo amor, carinho, admiração e apoio nesta importante etapa da minha vida. Agradeço por estar sempre ao meu lado, acreditando em mim e na concretização deste trabalho. A você, exprimo o meu profundo sentimento de gratidão!

Ao meu orientador desta dissertação, o Professor Jairo Rocha, pela excelente orientação prestada e por ter transmitido a mim confiança nos momentos de insegurança. Exprimo-lhe aqui a minha gratidão.

Ao co-orientador Professor Thiago Machado, pela sua disponibilidade em ajudar, sendo sempre prestativo e atencioso, me auxiliando na compreensão do assunto através dos seus ensinamentos e de suas críticas construtivas.

Aos professores Antônio Boness e Ana Wyse, pelo companheirismo e pelo seu reconhecimento na dispensa de disciplinas previamente cursadas por mim, me proporcionando mais tempo de dedicação para a minha dissertação.

Aos professores Miguel Aroztegui e Lucidio Formiga, pela disposição de ouvir e sugerir alternativas para o aprimoramento desta pesquisa.

A meus amigos Raul, Éwerton, Bruno, Neto e Manoel Messias pelos sábios conselhos, brincadeiras e também pelo companheirismo que tiveram ao longo desta caminhada.

Por fim, agradeço à Capes pelo apoio financeiro e a todos aqueles que contribuíram de forma direta e indireta para tornar realidade esta dissertação.

Resumo da Dissertação apresentada ao PPGMMC/CI/UFPB como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

O MÉTODO DAS SOLUÇÕES FUNDAMENTAIS APLICADO À RECONSTRUÇÃO DE FONTES CONCENTRADAS PARA PROBLEMAS ELÍPTICOS

Rômulo da Silva Lima

Março/2019

Orientadores: Jairo Rocha de Faria
Thiago José Machado

Programa: Modelagem Matemática e Computacional

O problema inverso estudado neste trabalho consiste em reconstruir uma fonte concentrada descrita por uma combinação linear finita de cargas puntuais do tipo delta de Dirac, tendo como base informações observadas na fronteira do domínio. Como exemplo de aplicações, podemos citar: identificação de hipocentros e epicentros de terremotos, conhecendo *a priori* os seus efeitos sobre a superfície da Terra; detecção de monopólos e dipolos em magnetoencefalografia e eletroencefalografia, auxiliando no diagnóstico de distúrbios cerebrais como tumores ou acidente vascular cerebral (AVC), por exemplo. Nesta dissertação, o problema inverso da reconstrução de fontes concentradas associado aos operadores elípticos de Laplace ou de Helmholtz é resolvido através de um problema de otimização. Em particular, o problema inverso é reformulado como um problema de minimização de um funcional de forma a ser minimizado com relação a um conjunto de fontes admissíveis. O Método das Soluções Fundamentais (MSF) é utilizado para resolver os problemas diretos auxiliares provenientes da reformulação do problema inverso, tendo em vista todas as vantagens deste método numérico sem malha, em comparação com técnicas de discretização do domínio, como o Método das Diferenças Finitas (MDF) e o Método dos Elementos Finitos (MEF), por exemplo. Além disso, o MSF é utilizado para representar as cargas puntuais que compõem a fonte concentrada, eliminando o ruído que é característico quando se usa discretização do domínio no algoritmo de reconstrução para a representação de fontes concentradas. Com os resultados numéricos obtidos, é possível comprovar a eficiência, eficácia e robustez do algoritmo de reconstrução proposto, mesmo considerando-se dados contaminados por ruídos.

Palavras-chave: Método das Soluções Fundamentais, Problemas Inversos de Fonte, Equação de Helmholtz, Equação de Poisson.

Abstract of Dissertation presented to PPGMMC/CI/UFPB as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

THE METHOD OF FUNDAMENTAL SOLUTIONS APPLIED TO POINTWISE SOURCES RECONSTRUCTION FOR ELLIPTIC PROBLEMS

Rômulo da Silva Lima

March/2019

Advisors: Jairo Rocha de Faria
Thiago José Machado

Program: Computational Mathematical Modelling

The inverse problem studied in this work is to reconstruct a concentrated source written by a finite linear combination of pointwise Dirac sources, based on information observed at the boundary of the domain. As an example of applications, we can point out: identification of hypocenters and epicenters of earthquakes, knowing a priori their effects on the Earth's surface; detection of monopoles and dipoles in magnetoencephalography and electroencephalography, aiding in the diagnosis of brain disorders such as tumors or stroke, for example. In this work, the inverse source problem associated with elliptical operators, such as the Laplace or Helmholtz operator, is solved through an optimization problem. In particular, the inverse source problem is reformulated as a minimization problem of a functional with respect to a set of admissible sources. The Method of Fundamental Solutions (MFS) is used to solve the direct auxiliary problems arising from the reformulation of the inverse problem, in view of all the advantages of this meshfree numerical method, as compared to domain discretization techniques, such as the Finite Difference Method (MDF) and the Finite Element Method (MEF), for example. In addition, the MFS is used to represent the pointwise Dirac sources that make up the concentrated source by a single point, eliminating the noise that is characteristic when using discretization of the domain in the reconstruction algorithm for the representation of the source. With the numerical results obtained, it is possible to prove the efficiency, effectiveness and robustness of the proposed reconstruction algorithm, even considering noisy data.

Keywords: Inverse Problems, Method of Fundamental Solution, Inverse Source problems, Helmholtz-type Equation, Poisson Equation.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Na presente dissertação, propomos um algoritmo para a solução do problema inverso de reconstrução de fontes em um domínio aberto e limitado, tendo como base os dados de Cauchy provenientes da fronteira do domínio de definição do problema. Como a unicidade é uma questão central para a reconstrução, as fontes consideradas neste trabalho são restritas a um conjunto especial de soluções admissíveis. Em particular, analisamos o caso em que a fonte é concentrada, em que propõe-se uma metodologia para obter sistematicamente cada um dos parâmetros que definem o termo fonte estudado.

A reconstrução de fontes concentradas é um problema sensível em relação ao ruído característico nas leituras realizadas sobre a fronteira do domínio. Em outras palavras, pequenas alterações nos dados de leitura podem ocasionar uma perda significativa na precisão da reconstrução da fonte. Dessa forma, para recuperar a estabilidade da reconstrução, propomos uma reformulação no modelo matemático que consiste em transformar o problema inverso em um problema de otimização. Neste caso, ao invés de lidar com um problema instável, essa reformulação matemática permite a manipulação de um funcional que tem como argumentos soluções de problemas diretos bem definidos, em que, através da análise de sensibilidade, tais problemas são utilizados no processo de reconstrução da solução do problema inverso.

A utilização de métodos numéricos que necessitem de discretizações ao longo do domínio também introduzem ruídos no algoritmo de reconstrução do termo fonte, uma vez que as localizações das cargas puntuais não coincidem necessariamente com o conjunto de pontos da malha utilizada. Nesse sentido, o Método das Soluções Fundamentais (MSF), que é um método numérico sem malha, é empregado para representar as cargas puntuais por meios dos denominados pontos fonte. Além disso, o MSF também é utilizado na etapa de validação do algoritmo proposto, em que é apresentada uma maneira de obtenção de leituras sintéticas na fronteira do domínio associadas a uma fonte concentrada, cujos parâmetros do termo fonte são

previamente conhecidos.

1.1 Problemas Diretos e Inversos

A teoria desenvolvida na área dos problemas diretos é bastante rica no que diz respeito aos procedimentos realizados na obtenção da solução devido ao intenso estudo que tem sido realizado ao longo dos últimos dois séculos. No contexto de equações diferenciais, o objetivo principal na formulação de um problema direto é a determinação da solução num certo domínio, a partir das condições de contorno e/ou iniciais conhecidas *a priori*.

De acordo com Kubo, 1988 [35], é necessário que se tenha o conhecimento de algumas informações *a priori* na resolução de um problema direto, tais como

(a) O domínio da solução do problema.

(b) A equação diferencial utilizada na modelagem.

(c) A condição de contorno em toda a fronteira do domínio e, no caso de problemas transientes, a condição inicial.

(d) As propriedades do material associadas à equação diferencial.

(e) Determinação das forças que atuam no domínio da solução do problema;

Quando todos os itens acima são satisfeitos, a solução pode ser calculada diretamente, seja através de técnicas matemáticas para se obter analiticamente a solução exata, seja através da utilização de métodos numéricos para se obter a solução aproximada do problema. Entretanto, quando uma ou mais das informações acima não for conhecida ou estiver incompleta, as análises diretas feitas anteriormente não podem ser aplicadas e, nesse caso, o problema é considerado como um problema inverso.

Os problemas inversos constituem uma área multidisciplinar que vem despertando o interesse de muitos pesquisadores nas últimas quatro décadas e suas aplicações surgem em diversos campos da ciência. Podemos citar, por exemplo, tomografias [2, 5, 42, 43], ultrassom e eletrocardiologia [23, 24, 26, 38, 48]; a identificação de anomalias ocultas na distribuição de densidade da Terra [3, 6, 30, 52]; os problemas de transferência de calor [45], ensaios não-destrutivos [7, 47]; estimativa de sinais discretos ruidosos degradados por meio de técnicas de processamento de imagem, podendo ser utilizado na restauração de imagens de satélites desfocadas, por exemplo [31]; entre outras aplicações.

Um problema é dito ser *correto*, *posto corretamente* ou *bem-posto* no sentido de Hadamard [27] se ele satisfaz as condições de existência, unicidade e dependência

contínua dos dados de entrada. Se uma ou mais dessas condições não for satisfeita, o problema é dito ser *posto incorretamente* ou *mal-posto* no sentido de Hadamard. Em geral, os problemas inversos não são bem-postos. Daremos a seguir uma definição mais formal sobre os conceitos acima citados.

Definição 1.1. *Sejam X e Y dois espaços normados e $\mathbb{L} : X \to Y$ um operador que relaciona X e Y através da equação*

$$\mathbb{L}[x] = y, \tag{1.1}$$

em que y é um dado e busca-se a solução x. A equação (1.1) é dita ser bem-posta se $\mathbb{L} : X \to Y$ é bijetivo e o operador inverso $\mathbb{L}^{-1} : Y \to X$ é contínuo. Caso contrário, a equação (1.1) é dita ser mal-posta.

Com base na definição acima, um problema mal-posto pode ocorrer de três formas distintas:

- Se o operador $\mathbb{L}$ não for sobrejetivo. Ou seja, para algum $y \in Y$, a equação (1.1) não tem solução.
- Se o operador $\mathbb{L}$ não for injetivo. Neste caso, isso significa que o problema (1.1) pode ter mais de uma solução.
- Se o operador inverso $\mathbb{L}^{-1}$ não for contínuo, a solução x da equação (1.1) não depende de forma contínua do dado de entrada y;

No caso mais simples, podemos pensar em um problema direto como sendo a determinação das raízes reais r_1 e r_2 de um polinômio P de grau 2, por exemplo. Neste caso, o problema inverso pode ser caracterizado como sendo o cálculo dos coeficientes de P, dadas as suas raízes reais. Claramente, esse problema não é bem-posto, uma vez que existem infinitos polinômios P_i com as raízes r_1 e r_2.

Em razão de sua interdisciplinaridade, a formulação e solução de um problema inverso envolve o conhecimento de diversos campos da matemática, ciências aplicadas e, além disso, são problemas mais difíceis de serem solucionados em comparação com os problemas diretos. Oleg Mikailivitch, o proponente de métodos inversos, afirma que um problema inverso pode ser definido pela identificação de causas, com base nas observações de seus efeitos [51]. Mas de acordo com [35], essa definição não se aplica em diversos casos e, portanto, uma definição mais racional seria considerá-los como o oposto dos problemas diretos.

Neste trabalho, baseado nos itens de (a)-(e), considera-se um dos seguintes problemas abaixo ou a combinação deles como sendo um problema inverso:

(i) A determinação do domínio, sua fronteira ou alguma fronteira interior desconhecida;

(ii) A inferência da equação diferencial que modela o problema;

(iii) A identificação da condição inicial e/ou de contorno;

(iv) A determinação das propriedades do material associadas à modelagem do problema;

(v) As forças que atuam dentro do domínio do problema.

Os problemas de reconstrução de fontes constituem um tipo de problema inverso que surge em diversas aplicações. Este tipo de problema considera equações diferenciais não homogêneas, em que é conhecida a fronteira do seu domínio a qual está sujeita a uma condição especificada. O termo fonte (ou as forças que atuam dentro do domínio) associado à equação diferencial não homogênea deve ser determinado.

1.2 Reconstrução de Fontes

Os problemas inversos de reconstrução de fontes é um tipo específico de problemas inversos com importantes aplicações nos mais variados campos da ciência. Em particular, podemos citar: detecção de epicentros e hipocentros de terremotos, conhecendo previamente os efeitos sobre a superfície da Terra [13, 34]; identificação de monopólos e dipolos em magnetoencefalografia e eletroencefalografia, que pode ajudar no diagnóstico de doenças cerebrais como tumores ou na prevenção de distúrbios cerebrais como o Acidente Vascular Cerebral (AVC) [21, 28]; reconstrução tomográfica para imagem molecular óptica, auxiliando no diagnóstico e tratamento de diversos tipos de câncer [32]; identificação de fontes de poluição em um rio, tendo como base a demanda química e biológica de oxigênio (DBO e DQO) [16, 17].

De acordo com Mamud [14], as principais classes de fontes mais encontradas na literatura são: fontes puntuais (representadas por combinações lineares de distribuições do tipo delta de Dirac); fontes características (dadas por combinações lineares de funções características); fontes acústicas (dadas pelo produto de uma função do número de onda por uma função da variável espacial); fontes escritas como um produto de funções de variáveis diferentes. Em [37], Machado fornece um método não-iterativo de segunda ordem para resolução do problema inverso do potencial considerando uma fonte concentrada. Em [18], esta fonte também é utilizada considerando a equação de Helmholtz, em que um resultado de unicidade é demonstrado. Já em [15], El Badia e Ha Doung consideram uma combinação de fontes mono e dipolares para a equação que modela EEG. Além disso, essa fonte também

é utilizada em [17] para a equação de difusão-dispersão-reação unidimensional no estudo de difusão de poluentes num rio, com base em medidas da demanda química e biológica de oxigênio (DQO e DBO).

Os estudos realizados em problemas inversos de reconstrução de fontes a partir dos dados de Cauchy mostram que são necessárias informações *a priori* acerca da fonte que se pretende reconstruir. Se nenhuma informação é previamente conhecida sobre a fonte, pode-se obter soluções diferentes de um problema inverso associadas a um mesmo dado de Cauchy. Dessa forma, este tipo de problema inverso é mal-posto, no sentido de Hadamard, pois não há garantia de unicidade da solução para fontes mais gerais. Por exemplo, El Badia e Nara [18] fornecem um algoritmo algébrico para a reconstrução de uma fonte monopolar associada ao problema de Helmholtz. Já El Badia e Ha Duong [15], num problema inverso de fonte para o modelo de EEG (electroencephalography), mostram um resultado de unicidade para fontes mono e dipolares, bem como a identificação delas consideradas num conjunto de fontes admissíveis.

Uma outra questão sobre os problemas inversos de fonte está relacionada à estabilidade do operador inverso. A instabilidade da solução causada pela não dependência contínua dos dados de entrada é de grande interesse no estudo de problemas inversos e diversos métodos de regularização foram desenvolvidos para contornar este problema como, por exemplo, técnicas de variação total ou regularização de Tikhonov [4, 25, 29, 39–41, 49]. Assim, com a restrição sobre a fonte de um problema inverso e com o uso de técnicas de regularização, é possível transformar um problema mal-posto em um bem-posto.

1.3 Proposta do Trabalho

No trabalho realizado em [44], a estratégia utilizada para reconstruir uma fonte concentrada em um domínio bidimensional foi reescrever o problema inverso como um problema de otimização em que, baseando-se no critério de Kohn-Vogelius [33], propõe-se a minimização de um funcional com relação a um conjunto de fontes admissíveis, resultando em um método computacional não iterativo para a resolução do problema inverso do potencial. Utilizando essa estratégia, o termo fonte foi construído com precisão, porém, obteve-se um custo computacional elevado em razão de alguns fatores, entre eles, a solução numérica de problemas diretos auxiliares advindos da metodologia empregada.

Neste trabalho, utiliza-se a mesma estratégia para a reconstrução das fontes concentradas em um problema modelado por equações do tipo Helmholtz e pela Equação de Poisson, em que emprega-se o Método das Soluções Fundamentais [12, 19, 20, 22] para o cálculo aproximado dos problemas diretos auxiliares provenientes da formu-

lação do problema de otimização. Além disso, considerando que o MSF é bem mais eficiente e de fácil implementação em comparação com técnicas de discretização do domínio, como o Método das Diferenças Finitas (MDF) e o Método dos Elementos Finitos (MEF), o MSF é mais apropriado para representar cargas puntuais.

1.4 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma. No Capítulo 2, encontra-se a formulação do problema, em que é introduzido o problema inverso e sua reformulação como um problema de otimização. No Capítulo 3, está a análise de sensibilidade, em que é extraído o algoritmo de reconstrução para a obtenção da solução do problema inverso. No Capítulo 4, é apresentado o Método das Soluções Fundamentais, bem como a sua utilização no algoritmo de reconstrução do problema inverso. No Capítulo 5, encontram-se os resultados numéricos obtidos considerando domínios bidimensionais e tridimensionais. Por fim, no Capítulo 6, está a conclusão do presente trabalho.

Capítulo 2

Formulação do Problema

Neste Capítulo, são introduzidos os problemas inversos associados às equações do tipo Helmholtz e de Poisson, em que este último é apresentado como um caso particular do primeiro. Além disso, são realizadas as reformulações de ambos os problemas. Apesar do problema de Poisson ser um caso particular de Helmholtz, tal relação não é tão simples quando tratamos o problema de otimização, pois o problema de Poisson apresenta algumas peculiaridades. Logo, tais problemas são reformulados em seções distintas.

2.1 Problema Inverso

Seja $\Omega \subset \mathbb{R}^n$, com $n = 2, 3$, um conjunto aberto e limitado, cuja fronteira Γ é um conjunto Lipschitz contínuo. Considere o seguinte problema de valor de contorno sobredeterminado:

$$\left\{ \begin{array}{rcl} (\lambda I - \Delta)u & = & b^* \\ u & = & u^* \\ -\partial_\eta u & = & q^* \end{array} \right\} \begin{array}{ll} \text{em} & \Omega, \\ \text{sobre} & \Gamma, \end{array} \tag{2.1}$$

em que $\lambda \in \mathbb{R}\backslash\{0\}$ e $\partial_\eta u$ representa a derivada normal externa de u. O problema inverso consiste em determinar a fonte $b^* : \Omega \to \mathbb{R}$ a partir de medições dos dados de Cauchy (u^*, q^*).

Se nenhuma hipótese for assumida sobre a fonte b^*, o problema (2.1) pode não ter solução única. Assim, é necessário admitir informações acerca da fonte em questão. Neste trabalho, assume-se que b^* pertence ao seguinte conjunto:

$$C_\delta(\Omega) = \left\{ b : \Omega \to \mathbb{R}; b(x) = \sum_{i=1}^{n} \alpha_i \delta(x - x_i) \right\}. \tag{2.2}$$

em que $n \in \mathbb{N}$ denota a quantidade de cargas puntuais e, para cada $i \in \{1, ..., n\}$,

$\alpha_i \in \mathbb{R}$ e $x_i \in \Omega$ denotam, respectivamente, as intensidades e as localizações das cargas. O conjunto $C_\delta(\Omega)$ é chamado de *conjunto de fontes admissíveis* e a fonte $b^* \in C_\delta(\Omega)$ pode ser escrita da seguinte forma:

$$b^*(x) = \sum_{i=1}^{m^*} \alpha_i^* \delta(x - x_i^*), \tag{2.3}$$

em que $\delta(x - x_i^*)$ é a distribuição de Dirac centrada no ponto x_i^*. A solução do problema (2.1) dentro do conjunto de fontes admissíveis $C_\delta(\Omega)$ existe e é única, conforme demonstrado em [18]. Além disso, a unicidade da solução é demonstrada em [37] para λ nulo. Neste último caso, o problema inverso é caracterizado como problema inverso do potencial, cuja formulação é dada por:

$$\left\{ \begin{array}{rcl} -\Delta u & = & b^* \\ u & = & u^* \\ -\partial_\eta u & = & q^* \end{array} \right\} \begin{array}{ll} \text{em} & \Omega, \\ \text{sobre} & \Gamma. \end{array} \tag{2.4}$$

2.2 Problema de Otimização

Em decorrência da presença de ruídos, muito frequente em problemas desse tipo, a reconstrução da solução ainda pode apresentar problemas de estabilidade. Isto é, pequenas variações nos dados de Cauchy (u^*, q^*) podem resultar na reconstrução de uma fonte com intensidades e localizações muito diferentes em relação a seus valores exatos. A ideia para contornar a dificuldade na estabilidade da reconstrução da solução consiste em reescrever o problema inverso (2.1) como um problema de otimização, em que um funcional de forma é minimizado com respeito ao conjunto de fontes admissíveis $C_\delta(\Omega)$. Como previamente mencionado, abordamos a reformulação matemática da equação de Poisson separadamente, tendo em vista as suas particularidades em relação as equações do tipo Helmholtz.

2.2.1 Equação do Tipo Helmholtz

Primeiramente, considera-se o seguinte funcional baseado no critério de Kohn-Vogelius [33]:

$$\mathcal{J}\left(u^D, u^N\right) = \frac{1}{2}\int_\Gamma (u^D - u^N)^2 dx, \tag{2.5}$$

em que u^D e u^N são soluções dos seguintes problemas auxiliares:

$$\left\{ \begin{array}{rcl} (\lambda I - \Delta)u^D & = & b_0 \quad \text{em} \quad \Omega, \\ u^D & = & u^* \quad \text{sobre} \quad \Gamma, \end{array} \right. \tag{2.6}$$

e

$$\begin{cases} (\lambda I - \Delta)u^N &= b_0 \quad \text{em} \quad \Omega, \\ -\partial_\eta u^N &= q^* \quad \text{sobre} \quad \Gamma, \end{cases} \tag{2.7}$$

em que b_0 é uma fonte arbitrária previamente definida dentro do conjunto de fontes admissíveis $C_\delta(\Omega)$. Ou seja, os problemas auxiliares (2.6) e (2.7) são problemas diretos cujas condições de contorno são provenientes do problema inverso (2.1). Sobre a regularidade das funções u^D e u^N, tem-se que elas pertencem ao espaço $W^{1,p}(\Omega)$, em que $p \in [1, 2)$ no caso 2D e $p \in [1, \frac{3}{2})$ no caso 3D. Em ambos os casos, o espaço $W^{1,p}(\Omega)$ está imerso em $L^2(\Omega)$ [11].

O funcional dado pela Eq.(2.5) está bem definido, uma vez que $(u^D - u^N) \in L^2(\Omega)$. Além disso, a proposição seguinte garante que este funcional se anula sobre a solução do problema inverso (2.1).

Proposição 2.1. *Sejam u^D e u^N soluções de (2.6) e (2.7), respectivamente. Se a fonte b_0 coincidir com a solução b^* do problema inverso (2.1), então $\mathcal{J}\left(u^D, u^N\right) = 0$.*

Demonstração. Defina $\Psi_1 = u - u^D$, em que u é proveniente do problema (2.1). Segue da hipótese que Ψ_1 é solução do seguinte problema

$$\begin{cases} (\lambda I - \Delta)\Psi_1 &= 0 \quad \text{em} \quad \Omega, \\ \Psi_1 &= 0 \quad \text{sobre} \quad \Gamma. \end{cases} \tag{2.8}$$

Como Ψ_1 se anula sobre Γ, pelo princípio do máximo para o problema de Dirichlet (Teorema 9.27 em [8]), segue que Ψ_1 é nula em todo domínio Ω e, portanto, $u = u^D$. Em contrapartida, definindo agora $\Psi_2 = u - u^N$, segue que

$$\begin{cases} (\lambda I - \Delta)\Psi_2 &= 0 \quad \text{em} \quad \Omega, \\ -\partial_\eta \Psi_2 &= 0 \quad \text{sobre} \quad \Gamma. \end{cases} \tag{2.9}$$

Calculando a formulação variacional, tomando a própria Ψ_2 como função teste, e usando a primeira identidade de Green, segue que

$$\int_\Omega \lambda \Psi_2^2 dx + \int_\Omega ||\nabla \Psi_2||^2 dx - \int_\Gamma \Psi_2 \partial_\eta \Psi_2 dx = 0. \tag{2.10}$$

Aplicando a condição de contorno de Neumann no problema da Eq.(2.9), temos que

$$0 = \int_\Omega \left(\lambda \Psi_2^2 + ||\nabla \Psi_2||^2\right) dx \geq \min\{1, \lambda\} \, ||\Psi_2||_{H^1(\Omega)} \geq 0. \tag{2.11}$$

Portanto, deve-se ter $||\Psi_2||_{H^1(\Omega)} = 0$, o que implica que $\Psi_2 = 0$ em Ω, encerrando a demonstração. □

2.2.2 Equação de Poisson

De maneira análoga ao que é feito com o problema inverso de fonte modelado pela equação do tipo Helmholtz, transformando-o em um problema de otimização, usamos o funcional de forma definido pela Eq.(2.5), em que agora as funções u^D e u^N são soluções dos seguintes problemas auxiliares:

$$\begin{cases} -\Delta u^D &= b_0 \quad \text{em} \quad \Omega, \\ u^D &= u^* \quad \text{sobre} \quad \partial\Omega, \end{cases} \tag{2.12}$$

e

$$\begin{cases} -\Delta u^N &= b_0 + c \quad \text{em} \quad \Omega, \\ -\partial_n u^N &= q^* \quad \text{sobre} \quad \partial\Omega, \\ \int_\Gamma u^N dx &= \int_\Gamma u^D dx. \end{cases} \tag{2.13}$$

Assim como no caso anterior, assume-se que a fonte b_0 é um elemento do conjunto de fontes admissíveis $C_\delta(\Omega)$. A constante c, adicionada ao termo fonte em (2.13), é necessária para garantir a compatibilidade do problema, isto é, para que o problema tenha solução, visto que são impostas condições de Neumann sobre a fronteira. Integrando no domínio Ω a equação diferencial no problema (2.13) e utilizando o teorema da divergência, segue que

$$c = \frac{1}{|\Omega|}\left(\int_\Gamma q^* dS - \int_\Omega b_0 dx\right), \tag{2.14}$$

em que $|\Omega|$ denota a área ou volume de Ω. Vale salientar que as soluções dos problemas (2.12) e (2.13) possuem as mesmas regularidades das soluções dos problemas (2.6) e (2.7) expostas anteriormente. Conforme demonstrado em [37], o funcional $\mathcal{J}$ se anula quando $b_0 = b^*$, em que b^* é solução do problema inverso (2.4).

Capítulo 3

Análise de Sensibilidade

Neste capítulo, realiza-se a análise da sensibilidade do funcional (2.5) com relação ao conjunto de fontes admissíveis. Isto é feito por meio do cálculo da variação do funcional, obtida através da perturbação da fonte b_0 dos problemas auxiliares. Após realizar a análise de sensibilidade, obtém-se um algoritmo que fornece a solução do problema de otimização, tanto para o problema do tipo Helmholtz quanto para o problema de Poisson.

3.1 Problemas Perturbados

Nesta seção, uma perturbação é realizada na fonte b_0 dos problemas auxiliares, com o intuito de avaliar os efeitos gerados sobre o funcional de Kohn-Vogelius. Essa perturbação é feita por meio da inserção de m cargas puntuais, com intensidades e localizações arbitrariamente definidas. Esse procedimento é feito considerando a equação do tipo Helmholtz e, logo em seguida, a equação de Poisson.

3.1.1 Equação do Tipo Helmholtz

Seja $b_\delta \in C_\delta(\Omega)$ a nova fonte perturbada, definida por:

$$b_\delta(x) = b_0(x) + \sum_{i=1}^{m} \alpha_i \delta_i(x), \tag{3.1}$$

em que $\delta_i(x) = \delta(x - x_i)$. Associados à fonte perturbada dada por (3.1), tem-se dois novos problemas perturbados, definidos por:

$$\begin{cases} (\lambda I - \Delta) u_\delta^D &= b_\delta \quad \text{em} \quad \Omega, \\ u_\delta^D &= u^* \quad \text{sobre} \quad \Gamma, \end{cases} \tag{3.2}$$

e

$$\begin{cases} (\lambda I - \Delta)u_\delta^N &= b_\delta \quad \text{em} \quad \Omega, \\ -\partial_\eta u_\delta^N &= q^* \quad \text{sobre} \quad \Gamma. \end{cases} \tag{3.3}$$

Assim, o funcional perturbado é dado por:

$$\mathcal{J}(u_\delta^D, u_\delta^N) = \frac{1}{2}\int_\Gamma (u_\delta^D - u_\delta^N)^2 dx. \tag{3.4}$$

A fim de avaliar a variação do funcional e analisar sua sensibilidade em relação às mudanças realizadas sobre a fonte, subtrai-se a Eq.(2.5) da Eq.(3.4) para se obter:

$$\mathcal{J}(u_\delta^D, u_\delta^N) - \mathcal{J}(u^D, u^N) = \frac{1}{2}\int_\Gamma \left[(u_\delta^D - u_\delta^N)^2 - (u^D - u^N)^2\right] dx. \tag{3.5}$$

Observe que na Eq.(3.5) não há uma relação explícita que evidencie a variação do funcional com os parâmetros m, α_i e x_i que definem cada elemento do conjunto $C_\delta(\Omega)$. Assim, são propostas duas expansões que estabelecem essa relação por meio da solução dos problemas auxiliares e perturbados.

Considere $v = u_\delta^D - u^D$, em que u_δ^D é solução de (3.2) e u^D é solução de (2.6). Segue que v é solução do seguinte problema de Dirichlet:

$$\begin{cases} (\lambda I - \Delta)v &= \displaystyle\sum_{i=1}^{m} \alpha_i \delta_i \quad \text{em} \quad \Omega, \\ v &= 0 \qquad\quad \text{sobre} \quad \Gamma. \end{cases} \tag{3.6}$$

Note que a função v depende de todos os parâmetros presentes na perturbação da fonte, ou seja, $v = v(m, \alpha_1, \ldots, \alpha_m, x_1, \ldots, x_m)$. Porém, seria interessante encontrar funções que dependessem apenas dos pontos de perturbação x_i. Para isto, escreve-se v como

$$v = \sum_{i=1}^{m} \alpha_i v_i. \tag{3.7}$$

Portanto, para $i = 1, 2, ..., m$, cada função v_i é solução do seguinte problema:

$$\begin{cases} (\lambda I - \Delta)v_i &= \delta_i \quad \text{em} \quad \Omega, \\ v_i &= 0 \quad \text{sobre} \quad \Gamma. \end{cases} \tag{3.8}$$

Claramente, cada função v_i depende apenas do ponto x_i. Desta maneira, a forma final da expansão que relaciona os problemas de Dirichlet (2.6) e (3.2) é dada por:

$$u_\delta^D = u^D + \sum_{i=1}^{m} \alpha_i v_i, \tag{3.9}$$

Com base na expansão (3.9), propõe-se agora uma expansão que relaciona os problemas (2.7) e (3.3), definida por

$$u_\delta^N = u^N + \sum_{i=1}^{m} \alpha_i (v_i + h_i), \tag{3.10}$$

em que cada função h_i é solução do seguinte problema:

$$\begin{cases} (\lambda I - \Delta) h_i &= 0 \quad \text{em} \quad \Omega, \\ -\partial_\eta h_i &= \partial_n v_i \quad \text{sobre} \quad \Gamma. \end{cases} \tag{3.11}$$

Observação 3.1. *Como a função h_i depende de v_i através da condição de contorno, pode-se dizer que há uma dependência implícita entre a função h_i e o ponto x_i. Observe ainda que as funções v_i e h_i não dependem da fonte arbitrária b_0 dos problemas auxiliares (2.6) e (2.7).*

A motivação para as duas expansões propostas para v_i e h_i baseia-se no fato do operador do tipo Helmholtz e de Poisson ser linear e que a diferença entre as fontes b_0 e b_δ resulta em um somatório de distribuições de Dirac.

3.1.2 Equação de Poisson

De maneira análoga, a fonte perturbada b_δ dos problemas auxiliares (2.12) e (2.13) é definida pela Eq.(3.1), gerando dois novos problemas perturbados, dados por:

$$\begin{cases} -\Delta u_\delta^D &= b_\delta \quad \text{em} \quad \Omega, \\ u_\delta^D &= u^* \quad \text{sobre} \quad \Gamma, \end{cases} \tag{3.12}$$

e

$$\begin{cases} -\Delta u_\delta^N &= b_\delta + c_\delta \quad \text{em} \quad \Omega, \\ -\partial_\eta u_\delta^N &= q^* \quad \text{sobre} \quad \Gamma, \\ \displaystyle\int_{\partial\Omega} u_\delta^N dx &= \displaystyle\int_{\partial\Omega} u_\delta^D dx, \end{cases} \tag{3.13}$$

em que a constante de compatibilidade c_δ é dada por:

$$c_\delta = c - \frac{1}{|\Omega|} \sum_{i=1}^{m} \alpha_i. \tag{3.14}$$

Assim como no caso anterior, a variação do funcional dada pela Eq.(3.5) não traz uma relação explícita com os parâmetros m, α_i e x_i que definem cada elemento do conjunto $C_\delta(\Omega)$. Logo, são usadas as expansões propostas pelas Equações (3.9) e (3.10) a fim de obter essa relação, em que agora as funções v_i e h_i são soluções dos seguintes problemas auxiliares:

$$\begin{cases} -\Delta v_i &= \delta_i \quad \text{em} \quad \Omega, \\ v_i &= 0 \quad \text{sobre} \quad \Gamma, \end{cases} \tag{3.15}$$

e

$$\begin{cases} -\Delta h_i &= \dfrac{1}{|\Omega|} \quad \text{em} \quad \Omega, \\ -\partial_n h_i &= \partial_n v_i \quad \text{sobre} \quad \Gamma, \\ \displaystyle\int_{\partial\Omega} h_i dx &= 0. \end{cases} \tag{3.16}$$

Na seção seguinte, são calculados os efeitos gerados na variação do funcional com a perturbação realizada sobre a fonte b_0. Tal análise é válida tanto para as equações do tipo Helmholtz, quanto para as equações de Poisson, uma vez que foi adotado o mesmo funcional e as mesmas expansões em ambos os casos.

3.2 Variação do Funcional de Forma

Com as expansões dadas pelas Eq.(3.9) e Eq.(3.10) e com as formulações dos novos problemas diretos auxiliares associados às funções v_i e h_i decorrentes das expansões propostas, é possível estabelecer uma relação explícita da variação do funcional (3.5) com os parâmetros m, α_i e x_i que definem os elementos do conjunto de soluções admissíveis $C_\delta(\Omega)$.

Desenvolvendo-se o integrando na Eq.(3.5) e reagrupando os termos, tem-se que:

$$\begin{aligned} \mathcal{J}(u_\delta^D, u_\delta^N) - \mathcal{J}(u^D, u^N) &= \frac{1}{2}\int_\Gamma \left(2u^D - 2u^N - \sum_{i=1}^m \alpha_i h_i\right)\left(-\sum_{i=1}^m \alpha_i h_i\right) dx \\ &= -\int_\Gamma \sum_{i=1}^m \alpha_i h_i (u^D - u^N)\, dx + \frac{1}{2}\int_\Gamma \left(\sum_{i=1}^m \alpha_i h_i\right)^2 dx. \end{aligned} \tag{3.17}$$

Note que a variação do funcional depende de forma explícita da quantidade de cargas puntuais m e das intensidades α_i. Além do mais, a variação do funcional depende implicitamente das localizações x_i, consoante a Observação 3.1. Com isso, pode-se escrever a variação do funcional como uma função dos parâmetros que caracterizam a fonte, isto é:

$$\mathcal{J}(u_\delta^D, u_\delta^N) - \mathcal{J}(u^D, u^N) = J(m, \boldsymbol{\alpha}, \boldsymbol{\xi}), \tag{3.18}$$

em que $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_m)$ e $\boldsymbol{\xi} = (x_1, x_2, ..., x_m)$. A Eq.(3.17) pode ser reescrita da seguinte forma:

$$J(m, \boldsymbol{\alpha}, \boldsymbol{\xi}) = -\langle \boldsymbol{d}, \boldsymbol{\alpha}\rangle + \frac{1}{2}\langle \boldsymbol{\alpha}, H\boldsymbol{\alpha}\rangle, \tag{3.19}$$

em que as entradas da matriz $H \in \mathbb{R}^{m\times m}$ e do vetor $\boldsymbol{d} \in \mathbb{R}^m$ são dadas por:

$$H_{ij} = \int_\Gamma h_i h_j dx \qquad \text{e} \qquad \boldsymbol{d}_i = \int_\Gamma h_i(u^D - u^N)dx. \tag{3.20}$$

Observe que a função J é uma forma quadrática com relação à variável $\boldsymbol{\alpha}$. Logo, desde que a matriz H é simétrica, o valor de $\widehat{\boldsymbol{\alpha}}$ que minimiza J é obtido através da seguinte equação:

$$\langle D_{\boldsymbol{\alpha}} J(m, \widehat{\boldsymbol{\alpha}}, \boldsymbol{\xi}), \boldsymbol{\beta}\rangle = 0, \forall \boldsymbol{\beta} \in \mathbb{R}^m. \tag{3.21}$$

Resolvendo a equação acima, tem-se o seguinte sistema matricial:

$$H\widehat{\boldsymbol{\alpha}} = \boldsymbol{d}. \tag{3.22}$$

Da Observação 3.1, segue que a matriz H e o vetor $\boldsymbol{d}$, definidos em (3.20), dependem do vetor de localizações $\boldsymbol{\xi}$. Consequentemente, a solução $\widehat{\boldsymbol{\alpha}}$ do sistema acima é, na verdade, uma função das localizações $\boldsymbol{\xi}$, ou seja, $\widehat{\boldsymbol{\alpha}} = \widehat{\boldsymbol{\alpha}}(\boldsymbol{\xi})$.

Seja X um conjunto obtido através de uma discretização arbitrária de Ω, que é denominado conjunto de localizações admissíveis. Sobre este conjunto, realiza-se uma busca combinatória a fim de obter o vetor de localizações que minimiza o funcional (3.17). Em outras palavras, o vetor de localizações ótimas $\boldsymbol{\xi}^\star$ é solução do seguinte problema de minimização:

$$\boldsymbol{\xi}^\star = \underset{\boldsymbol{\xi}\in X}{argmin} \left\{ J(m, \widehat{\boldsymbol{\alpha}}(\boldsymbol{\xi}), \boldsymbol{\xi}) = -\frac{1}{2}\langle \widehat{\boldsymbol{\alpha}}(\boldsymbol{\xi}), \boldsymbol{d}\rangle \right\}. \tag{3.23}$$

Em seguida, obtém-se imediatamente o vetor de intensidades ótimas $\boldsymbol{\alpha}^\star$, uma vez que este vetor é a solução do sistema (3.22) sobre o vetor de localizações ótimas, ou seja, $\boldsymbol{\alpha}^\star = \widehat{\boldsymbol{\alpha}}(\boldsymbol{\xi}^\star)$. Portanto, um vetor de localizações ótimas $\boldsymbol{\xi}^\star$ e um vetor de intensidades ótimas $\boldsymbol{\alpha}^\star$ podem ser determinados para cada valor de m, isto é, o problema de otimização tem uma solução diferente para cada m. A obtenção da quantidade correta de cargas puntuais m^* para um dado par de Cauchy (u^*, q^*) e os demais parâmetros ótimos são discutidos com mais detalhes na próxima seção.

3.3 Algoritmo de Busca da Solução Ótima

Nesta seção, discute-se como obter a quantidade correta de cargas puntuais m^* para um dado par de Cauchy (u^*, q^*) com suas respectivas intensidades $\boldsymbol{\alpha}^\star$ e localizações $\boldsymbol{\xi}^\star$ ótimas. Vale salientar que o conjunto discreto $X \subset \Omega$ definido na seção anterior é preestabelecido e, do ponto de vista do custo computacional, tem forte influência na eficiência da busca pela solução ótima, visto que uma busca combinatória é realizada sobre tal conjunto para o cálculo que minimiza a variação do funcional. Supõe-se que a solução do problema inverso é uma fonte com pelo menos uma carga puntual.

Inicialmente, supõe-se que há uma fonte com uma carga a ser reconstruída. Dessa forma, para cada ponto $p_i \in X$, resolve-se um sistema $H\widehat{\boldsymbol{\alpha}} = \boldsymbol{d}$, em que $H, \boldsymbol{d} \in \mathbb{R}$. O vetor ótimo de localizações $\boldsymbol{\xi}^\star$ associado a $m = 1$ é encontrado através do problema (3.23). Observe que o vetor ótimo de intensidades $\boldsymbol{\alpha}^\star$ associado a $m = 1$ também está determinado por $\boldsymbol{\xi}^\star$, pois $\boldsymbol{\alpha}^\star = \widehat{\boldsymbol{\alpha}}(\boldsymbol{\xi}^\star)$.

Em seguida, supõe-se que há uma fonte com duas cargas a serem reconstruídas. Dessa forma, para cada par de pontos $(p_i, p_j) \in X$, com $i \neq j$, resolve-se um sistema $H\widehat{\boldsymbol{\alpha}} = \boldsymbol{d}$, em que $H \in \mathbb{R}^{2\times 2}$ e $\boldsymbol{d} \in \mathbb{R}^2$. O vetor ótimo de localizações $\boldsymbol{\xi}^\star$ associado a $m = 2$ é determinado através do problema (3.23) e, consequentemente, o vetor ótimo de intensidades $\boldsymbol{\alpha}^\star$ também está determinado. Caso a solução do problema seja uma fonte com duas cargas puntuais, o cálculo do vetor ótimo $\boldsymbol{\alpha}^\star$ associado a $m = 3$ produzirá uma intensidade desprezível em relação às demais intensidades e, portanto, o algoritmo finalizará com os vetores ótimos $\boldsymbol{\xi}^\star$ e $\boldsymbol{\alpha}^\star$ associados a $m^* = 2$. Caso a solução do problema seja uma fonte com mais de duas cargas puntuais, o algoritmo seguirá calculando os vetores ótimos de intensidades e localizações para cada $m > 2$.

De maneira geral, quando houver k cargas puntuais a serem reconstruídas, isto é, para $m = k$, resolve-se um sistema $H\widehat{\boldsymbol{\alpha}} = \boldsymbol{d}$, em que $H \in \mathbb{R}^{k\times k}$ e $\boldsymbol{d} \in \mathbb{R}^k$. Se o vetor ótimo $\boldsymbol{\alpha}^\star$ associado a $m = k$ contiver uma entrada desprezível em relação às demais, então a solução ótima do problema será $\boldsymbol{\xi}^\star$ e $\boldsymbol{\alpha}^\star$ associados a $m^* = k - 1$.

Capítulo 4

Método das Soluções Fundamentais

O Método das Soluções Fundamentais é um método numérico sem malha que tem sido bastante utilizado em problemas de valores de contorno (PVC) quando a solução fundamental da equação diferencial é conhecida. A ideia geral do MSF consiste em aproximar a solução por uma combinação linear de soluções fundamentais, considerando um conjunto de pontos distribuídos fora do domínio de definição do problema. Em seguida, apresentam-se algumas definições importantes para uma melhor compreensão sobre a aplicabilidade do MSF.

Definição 4.1. *Seja $\mathbb{L}$ um operador linear diferencial. Dizemos que $\Phi(x, P)$ é uma solução fundamental para o operador $\mathbb{L}$ se,*

$$\mathbb{L}[\Phi(x, P)] = \delta(x, P),$$

em que δ é a distribuição delta de Dirac centrada no ponto P. Neste caso, a função $\Phi(x, P)$ está definida em $\mathbb{R}^n$, exceto em P, em que P é chamado de ponto singular ou singularidade para a solução fundamental Φ.

Definição 4.2. *Seja $\Omega \subset \mathbb{R}^n$ um conjunto aberto e limitado, cuja fronteira é denotada por Γ. Considere o seguinte problema:*

$$\mathbb{L}[u(x)] = 0, x \in \Omega, \tag{4.1}$$

sujeito a uma condição de contorno dada por:

$$\mathcal{B}[u(x)] = 0, x \in \Gamma, \tag{4.2}$$

em que $\mathbb{L}$ é um operador linear diferencial e $\mathcal{B}$ denota condição de contorno. A solução aproximada $\tilde{u}(x)$ utilizando o Método das Soluções Fundamentais para o problema (4.1)-(4.2) é dada pela combinação linear:

$$\tilde{u}(x) = \sum_{i=1}^{N} c_i \Phi(x, P_i),\ x \in \overline{\Omega}, \tag{4.3}$$

em que P_i são as singularidades da solução fundamental Φ consideradas fora de $\overline{\Omega}$. Os coeficientes c_i da Eq.(4.3), com $i = 1, ..., N$, podem ser determinados pela Eq.(4.2), isto é, usando o método da colocação sobre a fronteira Γ e resolvendo-se o sistema linear associado.

Em geral, o conjunto de pontos singulares P_i, no contexto do MSF, é chamado de *conjunto de pontos fonte* e pertence à fronteira $\widehat{\Gamma}$ de um conjunto $\widehat{\Omega}$, em que $\Omega \subset \widehat{\Omega}$, conforme ilustra a Figura 4.1. O conjunto de pontos x_i sobre a fronteira Γ, com $i = 1, ..., M$, é chamado de *conjunto de pontos de colocação*. A fronteira $\widehat{\Gamma}$ é uma *fronteira fictícia*, enquanto a fronteira Γ é a *fronteira física*.

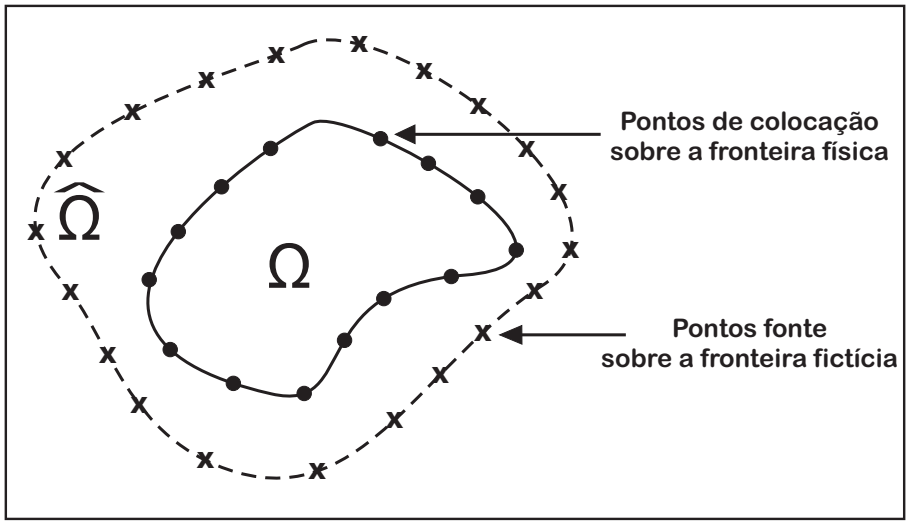

Figura 4.1: Representação gráfica da aplicação do MSF

Além da necessidade de conhecer a solução fundamental associada ao operador diferencial, outra desvantagem do MSF está relacionada ao fato de que o número de pontos fonte e suas respectivas localizações devem ser preestabelecidos, o que adiciona um certo grau de arbitrariedade ao método. Entretanto, usaremos esta arbitrariedade a favor do número λ para a equação do tipo Helmholtz. Mais especificamente, ajustaremos a fronteira fictícia do MSF para um dado λ de modo que a solução aproximada tenha uma melhor precisão na reconstrução. Vale salientar que o MSF clássico também pode ser adaptado para problemas não homogêneos, como mostra o trabalho realizado de Alves e Chen, 2005 [1], em que o método é utilizado para obter aproximações no problema de Poisson e de Helmholtz não homogêneo.

Uma das grandes vantagens da utilização do Método das Soluções Fundamentais, no presente contexto, é a representação de uma fonte concentrada com m cargas puntuais por m pontos fonte no interior do domínio.

Considere o seguinte PVC:

$$\begin{cases} (\lambda I - \Delta)u &= b \quad \text{em} \quad \Omega, \\ u &= u^* \quad \text{sobre} \quad \Gamma, \end{cases} \tag{4.4}$$

em que $\lambda \in \mathbb{R}$ e a fonte concentrada b é dada por:

$$b(x) = \sum_{i=1}^{m} \alpha_i \delta\left(x - y_i\right). \tag{4.5}$$

Com o objetivo de resolver numericamente o PVC (4.4) pelo MSF, é proposta a seguinte expansão:

$$u(x) = \sum_{i=1}^{N} c_i \phi(||x - P_i||) + \sum_{i=1}^{m} \alpha_i \phi(||x - y_i||), \tag{4.6}$$

em que P_i são os pontos fonte sobre a fronteira fictícia do PVC e $\phi(r)$ é a solução fundamental, dada por:

$$\phi(r) = \begin{cases} \dfrac{1}{2\pi} K_0(\sqrt{\lambda} r), & \text{se } \lambda > 0, \text{ (Helmholtz modificada)} \\ \dfrac{-1}{2\pi} log(r), & \text{se } \lambda = 0, \text{ (Laplace)} \\ \dfrac{i}{4} H_0^{(1)}(\sqrt{-\lambda} r), & \text{se } \lambda < 0, \text{ (Helmholtz)} \end{cases} \tag{4.7}$$

em $\mathbb{R}^2$ e

$$\phi(r) = \frac{e^{\sqrt{\lambda} r}}{4\pi r}, \tag{4.8}$$

em $\mathbb{R}^3$, em que $r = ||x||$, K_0 é a função de Bessel modificada de segundo tipo com ordem zero, e $H_0^{(1)}$ é a função de Hankel de primeiro tipo com ordem zero. Aplicando a condição de contorno de Dirichlet do PVC (4.4) na Eq.(4.6) e rearranjando os termos, obtém-se

$$\sum_{i=1}^{n} c_i \phi(||x_k - P_i||) = u^*(x_k) - \sum_{i=1}^{m} \alpha_i \phi(||x_k - y_i||), \tag{4.9}$$

em que x_k são os pontos de colocação sobre a fronteira física do PVC, com $i = 1, 2, ..., L$. Com isso, pode-se encontrar a solução aproximada de (4.4) resolvendo um sistema linear, em que a matriz desse sistema tem L linhas e N colunas. Em outras palavras, dado L pontos de colocação e N pontos fonte, a solução aproximada do PVC é dada por (4.6), em que cada coeficiente c_i é solução do sistema $Ac_i = g$, com a matriz A e o vetor g sendo dados por

$$A = \begin{bmatrix} \phi(\| x_1 - P_1 \|) & \cdots & \phi(\| x_1 - P_N \|) \\ \vdots & \ddots & \vdots \\ \phi(\| x_L - P_1 \|) & \cdots & \phi(\| x_L - P_N \|)) \end{bmatrix}_{L\times N} \tag{4.10}$$

$$g = \begin{bmatrix} u^*(x_1) - \sum_{i=1}^{m} \alpha_i \phi(||x_1 - y_i||) \\ \vdots \\ u^*(x_L) - \sum_{i=1}^{m} \alpha_i \phi(||x_L - y_i||) \end{bmatrix}_{L\times 1} \tag{4.11}$$

Com isso, o fluxo q^* associado à fonte concentrada b com m cargas puntuais pode ser facilmente obtido através da Eq.(4.6). Calculando a derivada normal da Eq.(4.6), obtém-se

$$\partial_\eta u(x) = \sum_{i=1}^{N} c_i \partial_\eta \phi(||x - P_i||) + \sum_{i=1}^{m} \alpha_i \partial_\eta \phi(||x - y_i||), \tag{4.12}$$

em que $\partial_\eta \phi(r)$ é dada por

$$\partial_\eta \phi(r) = \begin{cases} \dfrac{-\sqrt{\lambda}}{2\pi} K_1(\sqrt{\lambda} r) \dfrac{\partial r}{\partial \eta}, & \text{se } \lambda > 0, \\ \dfrac{-1}{2\pi r} \dfrac{\partial r}{\partial \eta}, & \text{se } \lambda = 0, \\ \dfrac{-i\sqrt{-\lambda}}{4} H_1^{(1)}(\sqrt{-\lambda} r) \dfrac{\partial r}{\partial \eta}, & \text{se } \lambda < 0, \end{cases} \tag{4.13}$$

em $\mathbb{R}^2$ e

$$\partial_\eta \phi(r) = \frac{e^{\sqrt{\lambda} r}(\sqrt{\lambda} r - 1)}{4\pi r^2} \frac{\partial r}{\partial \eta}, \tag{4.14}$$

em $\mathbb{R}^3$, em que K_1 é a função de Bessel modificada de segundo tipo com ordem um e $H_1^{(1)}$ é a função de Hankel de primeiro tipo com ordem um.

Portanto, os dados de Cauchy (u^*, q^*), gerados sinteticamente pelo MSF e associados à fonte b com m cargas puntuais dada pela Eq.(4.5), podem ser utilizados nos exemplos numéricos de reconstrução da fonte. Vale salientar que a solução numérica do problema direto associado a v_i apresentado no Capítulo 3 é um caso particular do problema (4.4), em que u^* é nulo e b é uma fonte concentrada com uma carga de intensidade $\alpha = 1$.

Capítulo 5

Resultados Numéricos

Neste capítulo, são apresentados os resultados numéricos obtidos com o algoritmo de reconstrução proposto. No primeiro momento, considera-se a equação do tipo Helmholtz (2D e 3D) para analisar o desempenho do algoritmo. No segundo momento, essa mesma análise é realizada considerando a equação de Poisson (2D), em que reconstrói-se o termo fonte no problema inverso do potencial.

A fonte $b_0 \in C_\delta(\Omega)$, caracterizada como o chute inicial, será adotada como nula, visto que não há influência dessa fonte na análise de sensibilidade. Em todos os resultados, cada carga puntual é representada por uma bola, com centro corresponde à localização e raio proporcional à sua intensidade.

As funções u^D, u^N, v_i, h_i foram obtidas utilizando o MSF clássico conforme explicado na Definição 4.2, uma vez que, considerando a equação do tipo Helmholtz, elas são soluções de problemas homogêneos. Entretanto, para a equação de Poisson, as referidas funções são soluções de problemas não homogêneos e, neste caso, utiliza-se a metodologia aplicada em [1].

Em relação às localizações dos pontos sobre as fronteiras física e fictícia do MSF, o experimento é calibrado para um dado $\lambda \neq 0$. Em particular, os pontos fonte são distribuídos uniformemente sobre a fronteira fictícia de um círculo de raio R centrado na origem. Além disso, o número de pontos de colocação sobre a fronteira física de Ω também é ajustado, como fica evidente na descrição do experimento.

Para computar os dados com ruído, o fluxo q^* é substituído pela relação $q^*_\mu = q^*(1 + \mu\gamma)$, em que γ é uma função que gera valores randomicamente entre -1 e 1 e μ é o nível percentual de ruído. Os experimentos numéricos foram desenvolvidos utilizando programação em MATLAB.

5.1 Problema de Poisson

Nesta seção, considerando a equação de Poisson, os experimentos numéricos tem como objetivos: analisar o comportamento do método com relação ao tamanho do conjunto de localizações admissíveis X; identificar o número correto de cargas puntuais para um dado par de Cauchy; analisar a reconstrução das cargas com dados poluído com ruído. Os experimentos foram realizados considerando um domínio bidimensional.

Adotamos um domínio Ω circular unitário centrado na origem. O par de dados de Cauchy (u^*, q^*) foram gerados computacionalmente utilizando o MSF, conforme explicado no Capítulo 4, em que $u^* = cos(\theta)$ sobre Γ, com $0 \leq \theta \leq 2\pi$. As soluções u^D, u^N, v_i e h_i foram obtidas pelo MSF utilizando $L = 30$ pontos de colocação e $N = 15$ pontos fonte, em que o valor do raio para a fronteira fictícia foi $R = 4$. Os pontos do conjunto de localizações admissíveis X (também chamado de *grid* de pontos) é gerado pela distribuição *sunflower seeds* [50] a fim de evitar o efeito indesejado de acumulação de pontos do *grid* em certas regiões. A Figura 5.1 ilustra um *grid* com 100 pontos. Para integração numérica, utiliza-se o método $\frac{1}{3}$ de Simpson com 25 pontos sobre a fronteira física para o cálculo da matriz H e do vetor d.

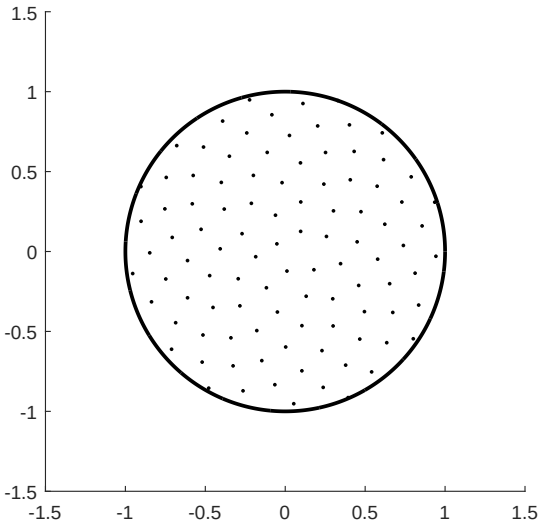

Figura 5.1: Simulação de um *grid* com 100 pontos seguindo a distribuição *sunflower seeds* num domínio circular.

Exemplo 1

Neste exemplo, analisamos a sensibilidade da reconstrução com relação ao tamanho do conjunto de localizações admissíveis X no problema de Poisson. Para isso, consideramos uma fonte com uma carga puntual localizada em $x^* = (0.40; 0.44)$ com

intensidade $\alpha^* = 5$. A reconstrução é feita utilizando 20, 100, 200 e 500 pontos no conjunto X. A Tabela (5.1) mostra os resultados obtidos na medida em que o *grid* é refinado. Observa-se que a precisão da reconstrução se eleva entre $\#X = 20$ e $\#X = 100$ pontos, e entre $\#X = 200$ e $\#X = 500$ pontos. Entretanto, nota-se que o aumento do número de pontos do conjunto X não eleva necessariamente a precisão da reconstrução, como mostram os resultados obtidos considerando $\#X = 100$ e $\#X = 200$ pontos. Percebe-se que a localização da carga, bem como a técnica utilizada para gerar o conjunto de pontos do *grid*, influencia na precisão da reconstrução. Nos Exemplos 2 e 3, considera-se um *grid* fixo com $\#X = 100$ pontos e que a localização de cada carga é um ponto dentro do *grid*, isto é, $x^* \in X$.

Tabela 5.1: Sensibilidade da reconstrução com relação ao número de elementos do conjunto de localizações admissíveis X para $x^* = (0.40; 0.44)$ e $\alpha^* = 5$.

$\#X$	20	100	200	500
$\boldsymbol{\xi}$	$(0.21; 0.28)$	$(0.40; 0.44)$	$(0.42; 0.52)$	$(0.39; 0.45)$
α	8.6	4.9	4.3	5.03
$\|\boldsymbol{\xi} - \boldsymbol{x}^*\|$	0.25	10^{-5}	0.078	0.012
$E_{rel}(\alpha)$	72 %	0.02 %	12.8 %	0.61 %

Exemplo 2

Neste exemplo, deseja-se obter o número correto de cargas puntuais no problema de Poisson para um dado par de Cauchy. Considera-se uma fonte com três cargas puntuais com localizações e intensidades dadas por $x_1^* = (0.40; 0.44)$, $x_2^* = (-0.57; 0.47)$, $x_3^* = (-0.45; -0.34)$, $\alpha_1^* = 5$, $\alpha_2^* = 10$ e $\alpha_3^* = 15$. A Figura 5.2 ilustra a solução exata a ser reconstruída.

O algoritmo é iniciado supondo com a reconstrução de uma fonte com uma carga puntual, isto é, $m = 1$. Em seguida, a reconstrução é realizada supondo um domínio com duas cargas puntuais ($m = 2$). Esse procedimento é ilustrado numericamente pela Tabela 5.2 e geometricamente pelas Figuras 5.3, 5.4, 5.5 e 5.6. Observe que o algoritmo produz uma quarta carga com intensidade desprezível para $m = 4$ ($\alpha_4 = 0.4968$). Além disso, as localizações das cargas com intensidades próximas dos seus valores exatos não mudam para $m = 3$ e $m = 4$. Assim, pode-se concluir que o número correto de cargas para esta fonte concentrada é $m^* = 3$.

Exemplo 3

Neste exemplo, analisa-se o comportamento da reconstrução com dados poluídos com ruído gaussiano branco. Considera-se uma fonte com três cargas puntuais com

Tabela 5.2: Reconstrução bidimensional da fonte para $m = 1, 2, 3$ e 4 no problema de Poisson.

m/α_i	α_1	α_2	α_3	α_4
$m = 1$	25.25	$- - - -$	$- - - -$	$- - - -$
$m = 2$	6.79	18.74	$- - - -$	$- - - -$
$m = 3$	5.4499	9.9075	15.6106	$- - - -$
$m = 4$	5.4624	10.1500	16.0132	0.4968

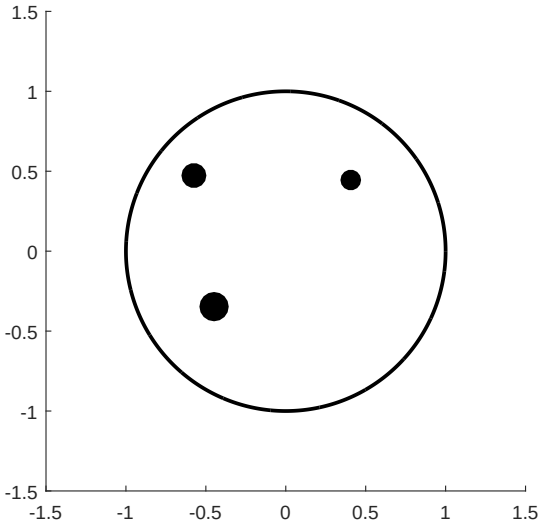

Figura 5.2: Solução bidimensional exata a ser reconstruída no problema de Poisson.

localizações $x_1^* = (0.72; 0.30)$, $x_2^* = (-0.38; 0.26)$, $x_3^* = (0.46; -0.54)$ e intensidade $\alpha^* = 8$ para todas as cargas. A Figura 5.7 ilustra a solução exata a ser reconstruída.

As localizações são reconstruídas de modo exato, com variações nas intensidades. A Tabela 5.3 mostra as variações obtidas nas intensidades. Observe que, para um nível de até $\mu = 40\%$ de ruído, os valores das intensidades reconstruídas não apresentam discrepâncias em relação aos seus valores exatos.

Tabela 5.3: Intensidade das cargas puntuais com inserção de ruído gaussiano branco em um domínio bidimensional no problema de Poisson.

μ_i/α_i	α_1	α_2	α_3
$\mu = 0\%$	7.9572	7.5215	7.7277
$\mu = 5\%$	7.9887	7.5047	7.8095
$\mu = 10\%$	7.9004	7.6603	7.6454
$\mu = 20\%$	7.8624	8.1605	7.5691
$\mu = 40\%$	7.3357	6.4975	7.3355

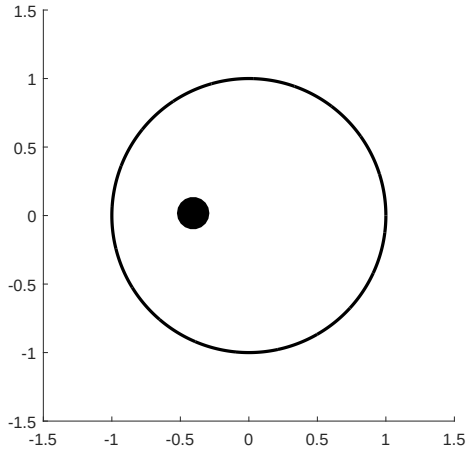

Figura 5.3: Reconstrução da fonte para $m = 1$ no problema de Poisson.

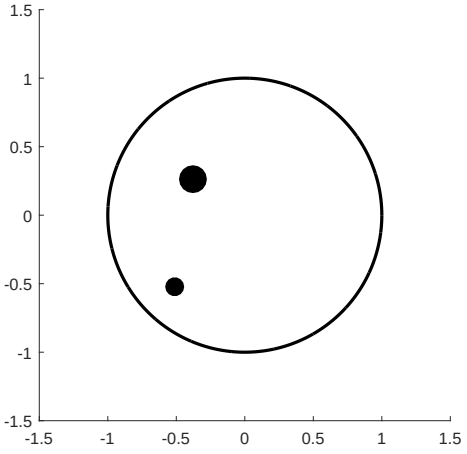

Figura 5.4: Reconstrução da fonte para $m = 2$ no problema de Poisson.

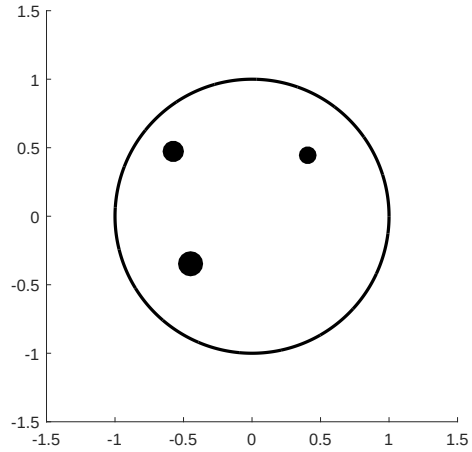

Figura 5.5: Reconstrução da fonte para $m = 3$ no problema de Poisson.

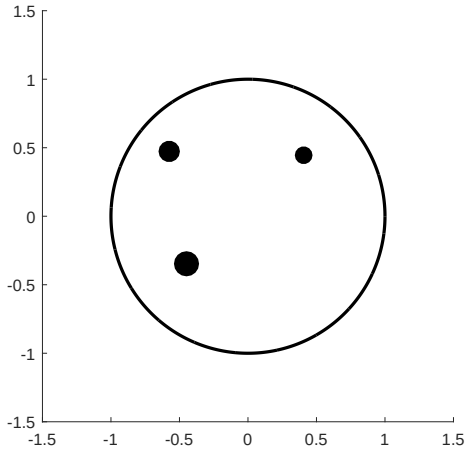

Figura 5.6: Reconstrução da fonte para $m = 4$ no problema de Poisson.

5.2 Equação de Helmholtz Modificada

Os exemplos numéricos apresentados nesta seção tem os seguintes objetivos:

- Analisar a fronteira fictícia do MSF que deve ser ajustada para um dado λ na equação de Helmholtz modificada;
- Mostrar a sensibilidade da reconstrução com relação ao número de pontos do conjunto de localizações admissíveis utilizado;
- Ilustrar o procedimento para identificar o número de cargas puntuais do problema inverso;
- Analisar o comportamento do método com relação a dados com ruído.

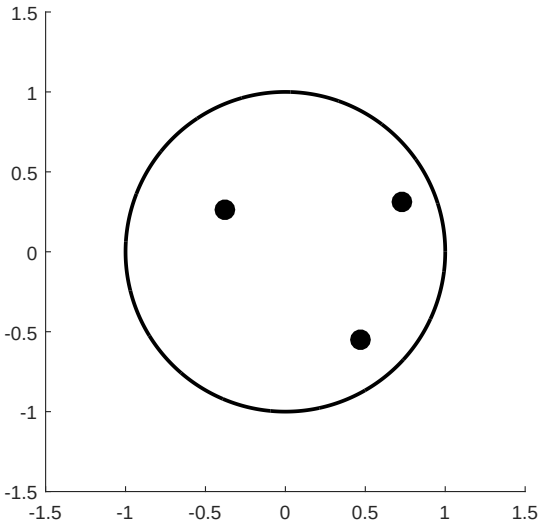

Figura 5.7: Solução exata bidimensional a ser reconstruída no problema de Poisson com dados ruidosos.

5.2.1 Domínio Bidimensional

Para o caso bidimensional, considera-se um domínio Ω circular de raio unitário centrado na origem. Os dados de Cauchy (u^*, q^*) associados à solução exata do problema inverso foram gerados utilizando o MSF, em que o valor de u^* na fronteira Γ foi escolhido como sendo $u^* = \cos(\theta)$, com $0 \leq \theta \leq 2\pi$.

O *grid* de pontos do conjunto de localizações admissíveis X é gerado novamente seguindo a distribuição *sunflower seeds* [50]. O cálculo da matriz H e do vetor $\boldsymbol{d}$ são realizados com a técnica de integração numérica $\frac{1}{3}$ de Simpson, usando-se apenas 25 pontos sobre a fronteira física Γ.

Exemplo 4

Este exemplo, de calibração do experimento numérico, tem o objetivo de encontrar o valor do raio R da fronteira fictícia do MSF, bem como o número de pontos fonte e de colocação que melhor se ajusta para um dado $\lambda > 0$ na equação de Helmholtz modificada. Para isso, utilizando inicialmente $\lambda_1 = +9.5$, fixa-se um *grid* de 100 pontos e, conforme R varia no intervalo $I = [1.1, 5]$, analisa-se o comportamento do método na reconstrução de uma carga puntual localizada em $x^* = (-0.11 + \Delta x; -0.22 - \Delta y)$ com intensidade $\alpha^* = 10$, nos casos em que $x^* \in X$, quando $\Delta x = \Delta y = 0$, e $x^* \notin X$ quando $\Delta x = \Delta y = 0.05$. Após diversos testes, obtiveram-se bons resultados utilizando $L_1 = 15$ pontos de colocação e $N_1 = 12$ pontos fonte. As Figuras 5.8 e 5.9 ilustram, respectivamente, o erro relativo de α quando $x^* \in X$ e $x^* \notin X$. Observe que na Figura 5.8, o erro relativo começa a estabilizar a partir de $R = 2.9$, e na Figura 5.9, o gráfico é estável a partir de $R = 1.2$, com erro em

torno de $10^{-8}\%$ quando $x^* \in X$ e próximo de 5% quando $x^* \notin X$. Em relação à localização da carga, tivemos $x = x^*$ com $x^* \in X$ e $||\boldsymbol{x} - \boldsymbol{x}^*|| = 0.07$ com $x^* \notin X$, para todo R no intervalo I, em que $||.||$ representa a distância euclidiana.

Fazendo essa mesma análise utilizando agora $\lambda_2 = +1$, analisa-se a reconstrução de uma carga localizada em $x^* = (-0.39 + \Delta x; 0.43 - \Delta y)$ com intensidade $\alpha^* = 20$. Utilizando $L_2 = 15$ pontos de colocação e $N_2 = 16$ pontos fonte para λ_2, o método obteve resultados mais satisfatórios. As Figuras 5.10 e 5.11 mostram, respectivamente, o erro relativo de α quando $x^* \in X$ e $x^* \notin X$. Note que a curva na Figura 5.11 começa a estabilizar a partir de $R = 1.5$, e na Figura 5.10, a curva se estabiliza com $R \in [1.1, 4.2]$. Em relação a localização da carga, tivemos os mesmos resultados obtidos com λ_1.

Nos Exemplos 5, 6 e 7, são usados os valores λ_1 e λ_2 no algoritmo de reconstrução, com raio $R = 3$ para ambos os valores, bem como o número de pontos fonte e de colocação utilizados neste exemplo para cada λ considerado.

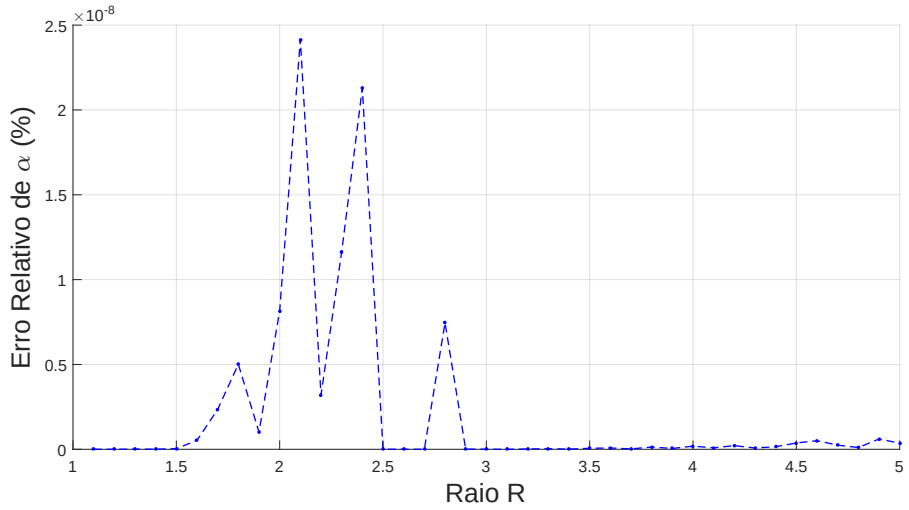

Figura 5.8: Erro relativo de α na reconstrução bidimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 10$ e $\lambda_1 = +9.5$, com $x \in X$.

Exemplo 5

Este exemplo objetiva analisar a sensibilidade do método de reconstrução com relação ao número de elementos do conjunto de localizações admissíveis X que é previamente definido nos testes. Inicialmente, considera-se a reconstrução de uma fonte com uma carga puntual localizada em $x^* = (0.25; -0.32)$, com a intensidade de $\alpha^* = 10$ e admitindo $\lambda_1 = +9.5$. Para as localizações admissíveis, levando-se em conta a natureza combinatória do algoritmo, considera-se X com 20, 100, 200 e 500 pontos, em que $x^* \notin X$. A Tabela 5.4 ilustra os resultados obtidos, em que $E_{rel}(\alpha)$ denota o erro relativo de α. Observe que, conforme é esperado, a precisão da

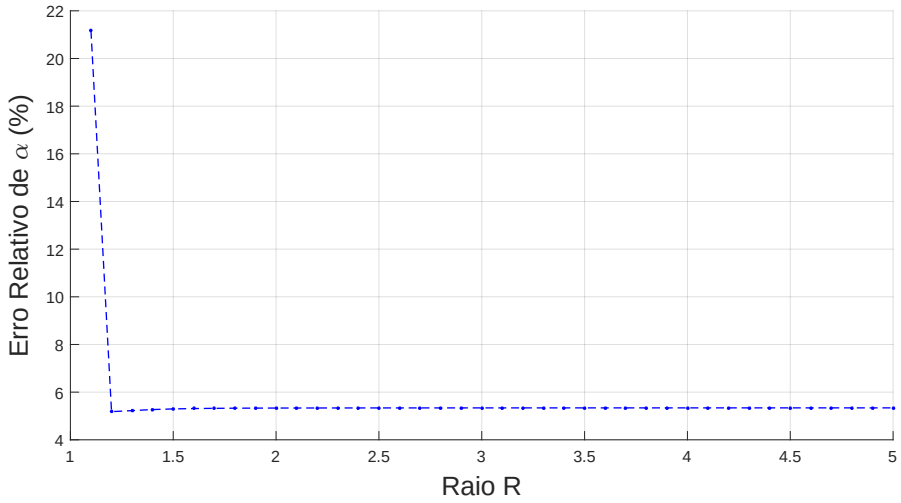

Figura 5.9: Erro relativo de α na reconstrução bidimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 10$ e $\lambda_1 = +9.5$, com $x \notin X$.

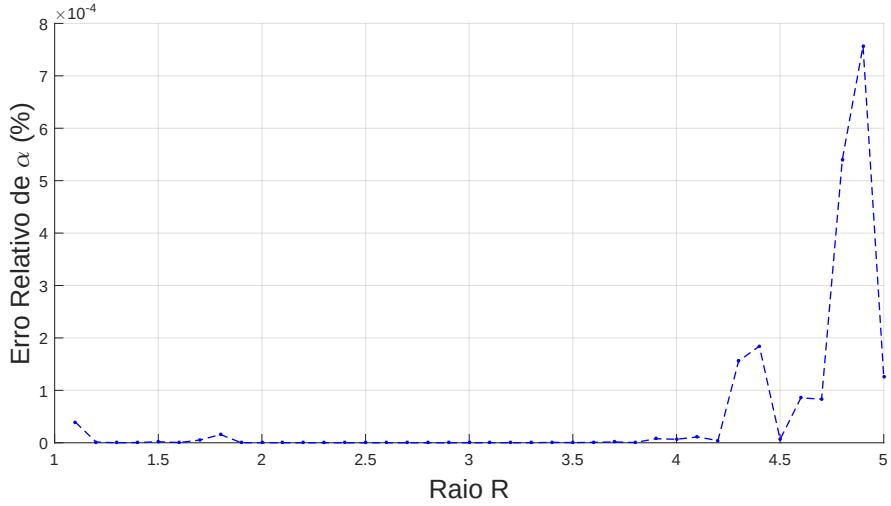

Figura 5.10: Erro relativo de α na reconstrução bidimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 20$ e $\lambda_2 = +1$, com $x \in X$.

reconstrução se eleva na medida em que o *grid* é refinado. No caso em que $x^* \in X$, o método reconstrói a localização exata e a intensidade com erro relativo de $10^{-7}\%$.

Fazendo a mesma análise, admitindo agora $\lambda_2 = +1$, analisa-se a reconstrução de uma carga localizada em $x^* = (0.14 + \Delta x; 0.48 - \Delta y)$ com intensidade $\alpha^* = 20$. A Tabela 5.5 mostra os resultados da reconstrução. Note que, mais uma vez, a solução aproximada fica mais próxima da solução exata na medida em que o *grid* é refinado. Para $x^* \in X$, a localização é reconstruída de modo exato e a intensidade com erro relativo de $10^{-6}\%$. Para os Exemplos 6 e 7, assume-se que $x^* \in X$ e, levando-se em conta o custo computacional, fixa-se X com 100 pontos.

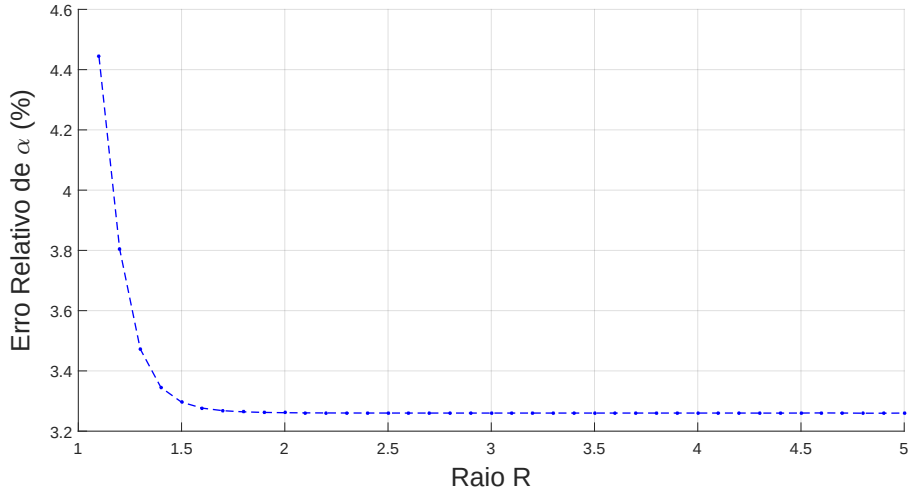

Figura 5.11: Erro relativo de α na reconstrução bidimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 20$ e $\lambda_2 = +1$, com $x \notin X$.

Tabela 5.4: Reconstrução bidimensional de uma carga puntual com diferentes conjuntos de localizações admissíveis para $x^* = (0.25; -0.32)$, $\alpha^* = 10$ e $\lambda_1 = +9.5$.

$\#X$	20	100	200	500
$\boldsymbol{\xi}$	(0.40; -0.25)	(0.29; -0.29)	(0.21; -0.32)	(0.24; -0.33)
α	7.8314	9.6264	10.3544	9.7806
$\|\boldsymbol{\xi} - \boldsymbol{x}^*\|$	0.1673	0.053	0.039	0.018
$E_{rel}(\alpha)$	21.6852 %	3.7353 %	3.5441 %	2.1939 %

Exemplo 6

Neste exemplo, reconstrói-se o número correto de cargas puntuais para uma dada fonte concentrada descrita pela Eq.(4.5). Inicialmente, considera-se uma fonte com três cargas puntuais, com localizações dadas por $x_1^* = (-0.39; 0.43)$, $x_2^* = (-0.45; -0.34)$ e $x_3^* = (0.57; 0.40)$, com a intensidade $\alpha^* = 6$ para todas as cargas, em que $\lambda_1 = +9.5$. A Figura 5.14 mostra a solução exata a ser reconstruída. O método de reconstrução, *a priori*, busca uma solução com uma carga puntual, isto é, $m = 1$. Em seguida, esse mesmo procedimento é realizado agora com $m = 2$, ou seja, o método busca outra solução com duas cargas puntuais. As Figuras 5.15, 5.16, 5.17 e 5.18 ilustram os resultados obtidos para cada valor de m. Observe que para $m = 4$, além do mesmo resultado para $m = 3$, obteve-se uma quarta carga com intensidade desprezível. As intensidades obtidas para $m = 4$ foram $\alpha_1 = 5.99997$, $\alpha_2 = 6.00042$, $\alpha_3 = 5.9989$ e $\alpha_4 = 0.0002$, com localizações exatas. Essa situação também pode ser vista na Figura 5.12 que ilustra o valor do funcional $\mathcal{J}$ dado pela Eq.(2.5) para cada valor de m. Note que para $m = 3$ e $m = 4$, o funcional atinge um valor próximo de zero, o que é esperado de acordo com a Proposição 2.1. Com isso, pode-se concluir que o número correto de cargas puntuais é $m^* = 3$.

Tabela 5.5: Reconstrução bidimensional de uma carga puntual com diferentes conjuntos de localizações admissíveis para $x^* = (0.14; 0.48)$, $\alpha^* = 20$ e $\lambda_2 = +1$.

$\#X$	20	100	200	500
$\boldsymbol{\xi}$	$(0.21; 0.28)$	$(0.09; 0.55)$	$(0.17; 0.43)$	$(0.09; 0.48)$
α	20.998	19.3881	20.2532	20.038
$\|\boldsymbol{\xi} - \boldsymbol{x}^*\|$	0.2107	0.086	0.0571	0.041
$E_{rel}(\alpha)$	4.99 %	3.059 %	1.266 %	0.1915 %

Repetindo o procedimento acima para $\lambda_2 = +1$, considera-se uma fonte com três cargas puntuais localizadas em $x_1^* = (0.45; -0.21)$, $x_2^* = (-0.47; -0.15)$ e $x_3^* = (0.02; 0.72)$, com intensidades $\alpha_1^* = 5$, $\alpha_2^* = 10$ e $\alpha_2^* = 15$. A Figura 5.19 mostra a solução exata a ser reconstruída. A Tabela 5.6 ilustra os resultados das intensidades obtidas para cada valor de m. Note que, para $m = 3$ e $m = 4$, existem três cargas com intensidades muito próximas. Além disso, a reconstrução para $m = 4$ produz uma quarta carga com intensidade desprezível. Essa situação também pode ser analisada com a Figura 5.13 que mostra os valores de $\mathcal{J}$ para cada m considerado. Logo, o número correto de cargas é $m^* = 3$.

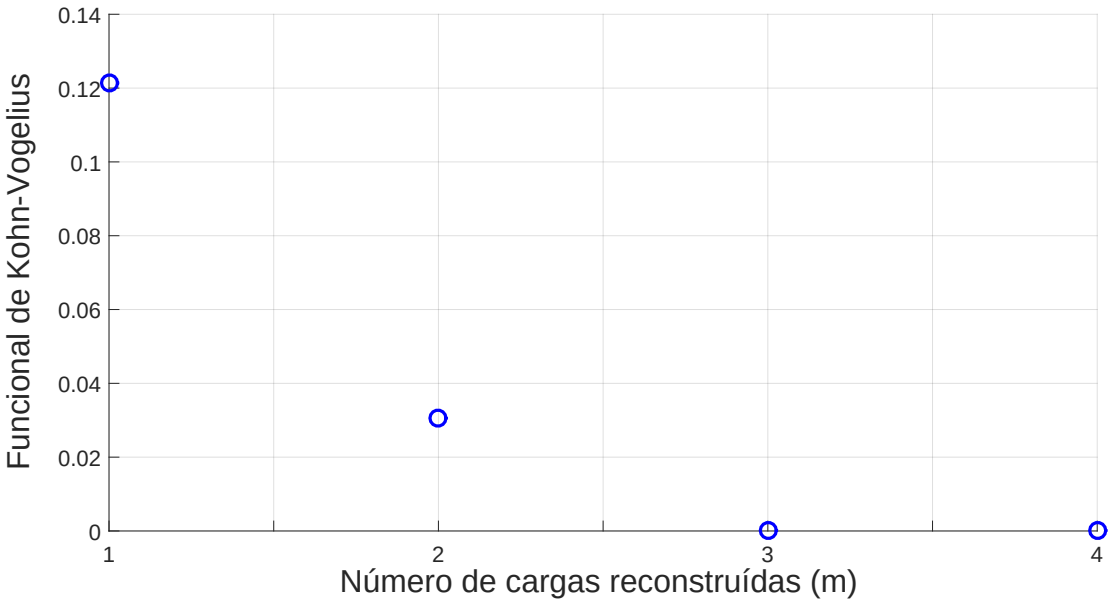

Figura 5.12: Valores do funcional de Kohn-Vogelius no problema de Helmholtz modificado para cada m considerado, com $\lambda_1 = +9.5$.

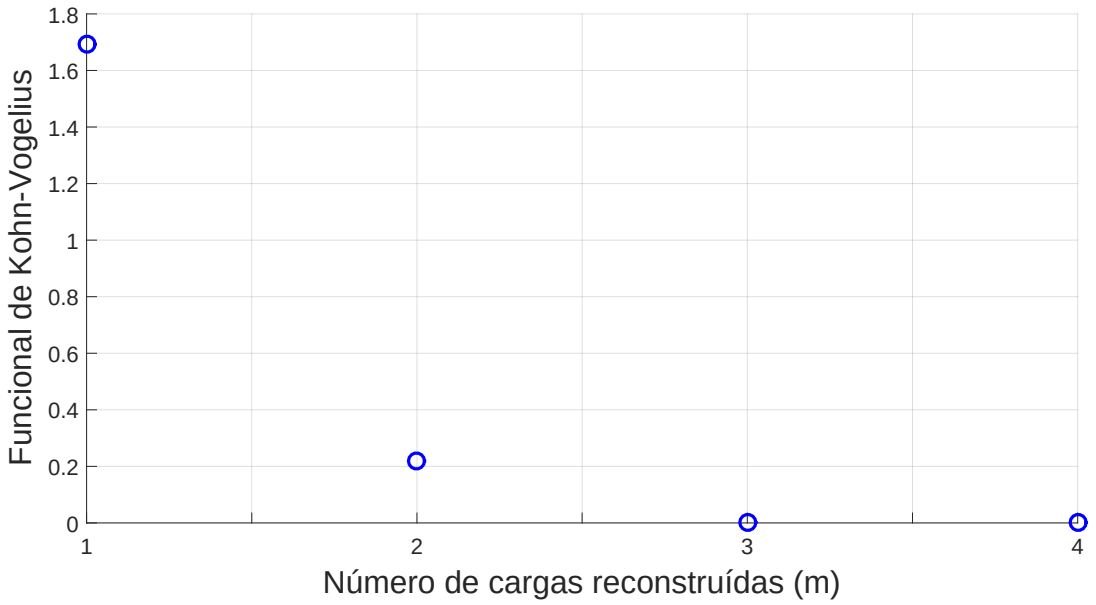

Figura 5.13: Valores do funcional de Kohn-Vogelius no problema de Helmholtz modificado para cada m considerado, com $\lambda_1 = +1.0$.

Tabela 5.6: Reconstrução bidimensional da fonte para $m = 1, 2, 3$ e 4, com $\lambda_2 = +1$.

m/α_i	α_1	α_2	α_3	α_4
$m = 1$	32.6703	$---$	$---$	$---$
$m = 2$	12.667	18.348	$---$	$---$
$m = 3$	4.99999995	10.00000006	14.99999996	$---$
$m = 4$	4.8456	9.7506	14.2615	1.2163

Exemplo 7

Neste exemplo, analisa-se o comportamento do método de reconstrução na presença de dados poluídos com ruído. Para isso, considera-se inicialmente a reconstrução de uma fonte concentrada com três cargas puntuais localizadas em $x_1^* = (0.72; 0.30)$, $x_2^* = (-0.38; 0.26)$ e $x_3^* = (0.46; -0.54)$ com as intensidades $\alpha_1^* = 5$, $\alpha_2^* = 10$ e $\alpha_3^* = 15$, supondo $\lambda_1 = +9.5$. A Figura 5.20 mostra a solução exata a ser reconstruída. O fluxo q^* é corrompido com os seguintes níveis de ruído: $\mu = 5\%$, $\mu = 10\%$, $\mu = 20\%$ e $\mu = 40\%$. As localizações das cargas são reconstruídas de modo exato apenas para os níveis de ruído $\mu = 0\%$, $\mu = 5\%$ e $\mu = 10\%$. Para $\mu = 20\%$, a carga 2 é reconstruída com uma distância de $||\boldsymbol{x_2} - \boldsymbol{x_2}^*|| = 0.286$, com localização exata para as cargas 1 e 3. Para $\mu = 40\%$, as cargas 2 e 3 obtiveram um erro na localização de $||\boldsymbol{x_2} - \boldsymbol{x_2}^*|| = 0.286$ e $||\boldsymbol{x_3} - \boldsymbol{x_3}^*|| = 0.1680$, com localização exata apenas para a carga 1. A Tabela 5.7 mostra os resultados obtidos das intensidades para os diferentes níveis de ruído. Podemos observar que, considerando um nível de até $\mu = 10\%$, não há uma discrepância demasiada em relação aos valores exatos das intensidades. Entretanto, para os níveis $\mu = 20\%$ e $\mu = 40\%$, as intensidades

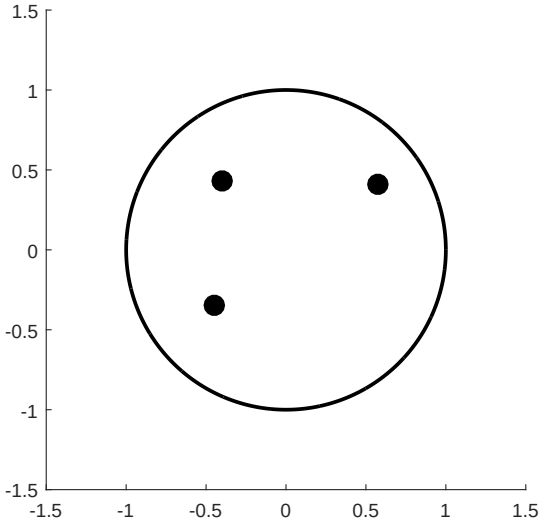

Figura 5.14: Solução bidimensional exata a ser reconstruída, com $\lambda_1 = +9.5$.

apresentaram valores com diferenças significativas em relação às intensidades exatas.

Fazendo essa mesma análise para $\lambda_2 = +1$, considera-se uma fonte com três cargas puntuais localizadas em $x_1^* = (-0.19; 0.47)$, $x_2^* = (0.65; -0.20)$ e $x_3^* = (-0.68; -0.44)$, com intensidade $\alpha^* = 5$ para todas as cargas. A Figura 5.21 ilustra a solução exata. As localizações são reconstruídas de modo exato. A Tabela 5.8 mostra os resultados obtidos das intensidades para os níveis de ruído preestabelecidos anteriormente. Note que, neste caso, a reconstrução obteve resultados satisfatórios até mesmo para os altos níveis de ruído adotados.

Tabela 5.7: Intensidade das cargas com inserção de ruído gaussiano branco em um domínio bidimensional para $\lambda_1 = +9.5$.

μ/α_i	α_1	α_2	α_3
$\mu = 0\%$	5.00000001	10.00000014	14.99999990
$\mu = 5\%$	4.9715	10.1585	15.125
$\mu = 10\%$	5.1051	11.1355	14.4492
$\mu = 20\%$	5.5509	16.0304	3.3066
$\mu = 40\%$	4.0149	17.772	4.0581

Tabela 5.8: Intensidade das cargas com inserção de ruído gaussiano branco em um domínio bidimensional para $\lambda_2 = +1$.

μ/α_i	α_1	α_2	α_3
$\mu = 0\%$	5.000000009	5.00000014	4.9999998
$\mu = 5\%$	4.7242	5.0814	4.9405
$\mu = 10\%$	5.2586	4.8435	4.9671
$\mu = 20\%$	5.5411	4.7146	4.8064
$\mu = 40\%$	4.2521	5.4626	5.6903

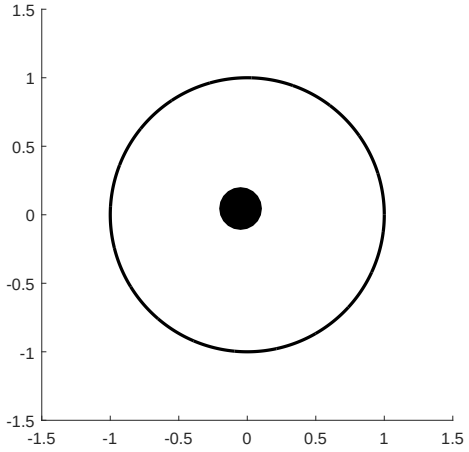

Figura 5.15: Reconstrução bidimensional da fonte para uma carga ($m = 1$).

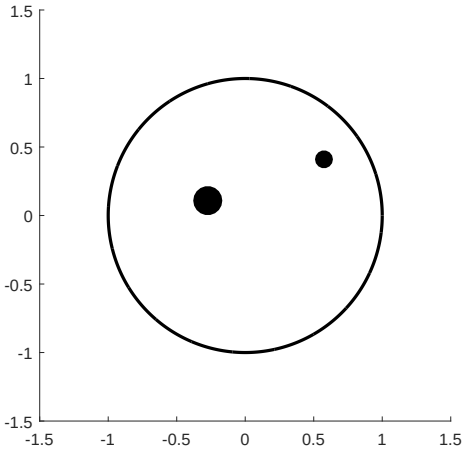

Figura 5.16: Reconstrução bidimensional da fonte para duas cargas ($m = 2$).

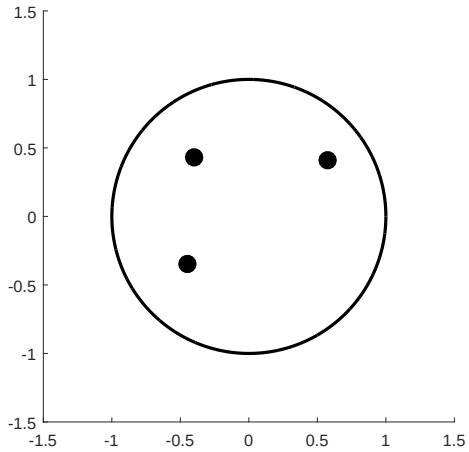

Figura 5.17: Reconstrução bidimensional da fonte para três cargas ($m = 3$).

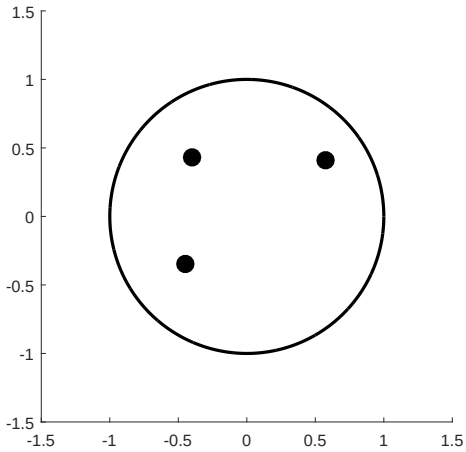

Figura 5.18: Reconstrução bidimensional da fonte para quatro cargas ($m = 4$).

5.2.2 Domínio Tridimensional

Para o caso tridimensional, o domínio Ω considerado é uma esfera unitária centrada na origem. O dado de Dirichlet prescrito na fronteira é adotado como nulo, isto é, $u^*(x) = 0, \forall x \in \Gamma$, em que o correspondente dado de Neumann q^* é gerado conforme descrito no Capítulo 4. Para a solução numérica dos problemas diretos auxiliares, os pontos fonte são distribuídos sobre a fronteira de uma esfera centrada na origem cujo raio R e o número de pontos é ajustado para um dado λ. Além disso, uma ajuste também é realizado com relação ao número de pontos de colocação sobre a fronteira de Ω, em que as implementações `sphere_cubed_point_num`, `sphere_cubed_points` e `sphere_cubed_points_face` [10] são utilizadas para gerar os pontos sobre a fronteira física e fictícia dos problemas diretos. A Figura 5.22 ilustra uma simulação da distribuição dos pontos fonte e de colocação utilizando um raio $R = 2$ para a fronteira fictícia, com 53 pontos em ambas as fronteiras. O *grid* de pontos do conjunto X é computado pelas implementações `grid_ball` e `grid_ball_count` [9].

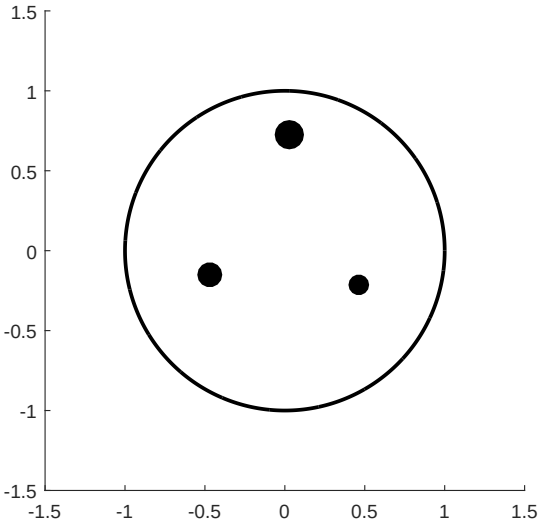

Figura 5.19: Solução bidimensional exata a ser reconstruída, com $\lambda_2 = +1$.

Levando em conta que a matriz H e o vetor $\boldsymbol{d}$ são calculados pela integral sobre a superfície de Ω, utiliza-se a implementação `getLebedevSphere` [46] que calcula a integral aproximada de uma função sobre a superfície da esfera unitária, centrada na origem, usando a quadratura de Lebedev [36]. O número de pontos para a integral de superfície utilizado foi de 434.

Exemplo 8

Assim como é feito no Exemplo 4, neste exemplo, ajusta-se o valor do raio R para um dado λ, bem como o número de pontos fonte e de colocação que é utilizado na reconstrução das cargas num domínio tridimensional. Para tanto, admitindo-se inicialmente $\lambda_1 = +9.5$, um *grid* de 117 pontos é fixado e considera-se a reconstrução de uma carga puntual com localização $x^* = (-0.66 + \Delta x; -0.33 - \Delta y; -0.33 + \Delta z)$ e intensidade $\alpha^* = 2$ para analisar o comportamento do método conforme o raio R varia, nos casos em que $x^* \in X$, quando $\Delta x = \Delta y = \Delta z = 0$, e $x^* \notin X$ quando $\Delta x = \Delta y = \Delta z = 0.05$. Utilizando $L = 26$ pontos de colocação e $N = 56$ pontos fonte, o método de reconstrução obteve resultados bastante satisfatórios. A Figura 5.23 mostra o erro relativo da intensidade α, em que $x^* \in X$. Já a Figura 5.24 ilustra o erro relativo quando $x^* \notin X$. Note que, a partir de $R = 1.5$ na Figura 5.24, o erro estabiliza-se em torno de 8%. Entretanto, o erro na Figura 5.23 é em torno de 10^{-7}% para $R \in [2.6, 4.3]$. A localização é reconstruída de modo exato quando $x^* \in X$ e com distância $||\boldsymbol{x} - \boldsymbol{x}^*|| = 0.0866$ quando $x^* \notin X$, para todo R no intervalo $[1.1, 5]$.

Fazendo essa mesma análise, admitindo agora $\lambda_2 = +1$, com o mesmo número de pontos fonte e de colocação anteriormente utilizados, considera-se a reconstrução de

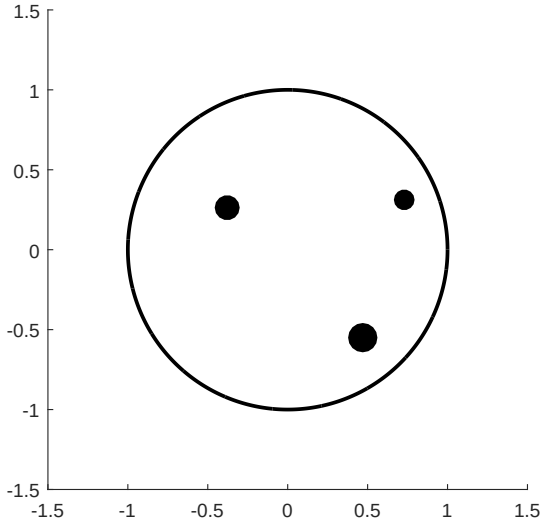

Figura 5.20: Solução bidimensional exata a ser reconstruída com dados ruidosos para $\lambda_1 = +9.5$.

uma carga puntual, cuja localização é dada por $x^* = (0.33 + \Delta x; -0.33 - \Delta y; 0.66 + \Delta z)$ e intensidade $\alpha^* = 4$. As Figuras 5.25 e 5.26 mostram, respectivamente, o erro relativo de α para $x^* \in X$ e $x^* \notin X$. Note que a curva na Figura 5.26 começa a estabilizar a partir de $R = 2$, com erro próximo de 12%. Já na Figura 5.25, obteve-se um erro em torno de $10^{-16}\%$ para $R \in [1.1, 3.4] \cup [4, 5]$. Com relação a localização da carga, tivemos os mesmos resultados obtidos com λ_1.

Portanto, nos Exemplos 7, 8 e 9, são usados os valores λ_1 e λ_2, com raios $R_1 = 3.5$ e $R_2 = 2.0$, respectivamente, bem como o mesmo número de pontos fonte e de colocação utilizados neste exemplo para cada λ considerado.

Exemplo 9

Neste exemplo, a sensibilidade do método de reconstrução é analisada em relação ao tamanho do *grid* (conjunto de localizações admissíveis X). Considera-se uma fonte tridimensional com uma carga puntual localizada em $x^* = (-0.25; 0; 0.35)$ e intensidade $\alpha^* = 10$, supondo $\lambda_1 = +9.5$. Para o conjunto X, os testes são realizados com $27, 93, 437, 799$ e 2403 pontos distribuídos no interior do domínio Ω, em que $x \notin X$. A Tabela 5.9 ilustra os resultados obtidos. Observe que, na medida em que o tamanho do *grid* aumenta, a reconstrução fica mais precisa. No caso em que $x^* \in X$, a localização é reconstruída de modo exato, em que o erro relativo de α está entre $10^{-7}\%$ e $10^{-4}\%$.

Fazendo a mesma análise supondo agora $\lambda_2 = +1$, considera-se a reconstrução de uma carga com localização $x^* = (-0.25; 0.34; 0.21)$ e intensidade $\alpha^* = 20$. A Tabela 5.10 mostra os resultados obtidos. Note que há uma melhora na precisão da localização da carga na medida em que tamanho do *grid* aumenta. Com relação à

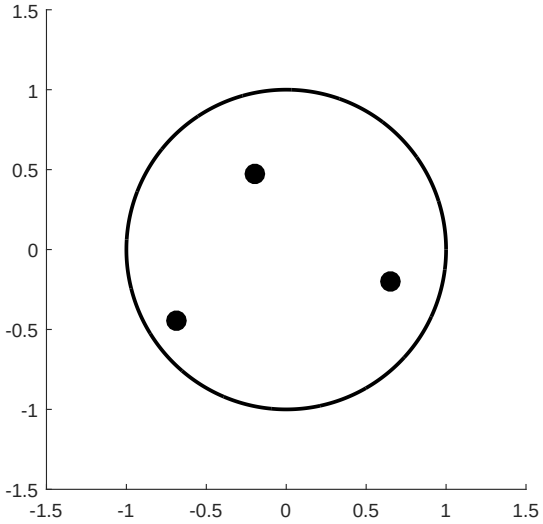

Figura 5.21: Solução bidimensional exata a ser reconstruída com dados ruidosos para $\lambda_2 = +1$.

intensidade, note que há uma perda de precisão entre 27 e 93 pontos, retomando um resultado mais preciso com 437 pontos, e entre 437 e 799 pontos, retomando com 2403 pontos um resultado melhor. Isso mostra que um *grid* mais refinado não resulta necessariamente em uma reconstrução mais precisa da intensidade, em que percebe-se que há uma dependência da localização exata da carga, bem como do tipo de distribuição utilizada para gerar o *grid* de pontos. Nos exemplos que se seguem, assumi-se que $x^* \in X$ e tamanho do *grid* é fixado em $\#X = 93$.

Tabela 5.9: Reconstrução tridimensional de uma carga puntual com diferentes conjuntos de localizações admissíveis para $x^* = (-0.25; 0; 0.35)$ e $\alpha^* = 10$, supondo $\lambda_1 = +9.5$.

$\#X$	27	93	437	799	2403
α	0.1065	8.2471	8.7231	11.0650	10.1616
$\|\boldsymbol{\xi} - \boldsymbol{x}^*\|$	0.5233	0.0950	0.0908	0.0778	0.0148
$E_{rel}(\alpha)$	98.9351 %	17.5293 %	12.7686 %	10.6497 %	1.6159 %

Tabela 5.10: Reconstrução tridimensional de uma carga puntual com diferentes conjuntos de localizações admissíveis para $x^* = (-0.25; 0.34; 0.21)$ e $\alpha^* = 20$, supondo $\lambda_2 = +1$.

$\#X$	27	93	437	799	2403
α	20.6707	18.6852	19.7835	20.2205	19.9250
$\|\boldsymbol{\xi} - \boldsymbol{x}^*\|$	0.4713	0.1649	0.1005	0.0863	0.0322
$E_{rel}(\alpha)$	3.3538 %	6.5735 %	1.0824 %	1.1029 %	0.3745 %

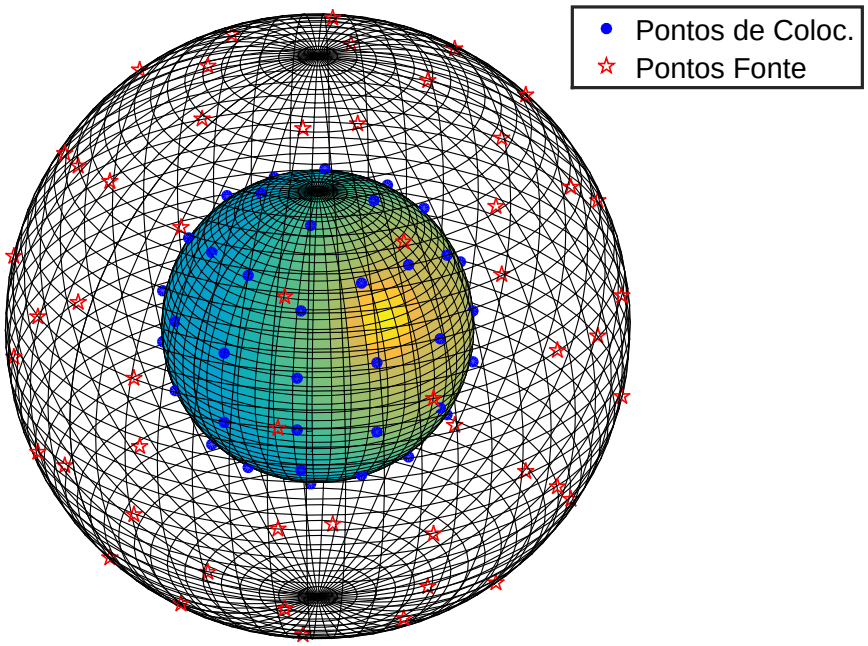

Figura 5.22: Simulação da distribuição de pontos fonte e de colocação sobre a fronteira física e fictícia.

Exemplo 10

Neste exemplo, identifica-se o número correto de cargas puntuais para um dado par (u^*, q^*) associado a uma fonte concentrada em um domínio tridimensional, em que o número de cargas puntuais é desconhecido *a priori*. Para isso, admitindo $\lambda_1 = +9.5$, considera-se uma fonte com três cargas puntuais localizadas em $x_1^* = (0; 0.34; 0.34)$, $x_2^* = (-0.34; 0; -0.34)$ e $x_3^* = (0.68; 0.34; -0.34)$, com intensidade $\alpha^* = 8$ para todas as cargas. Seguindo a mesma metodologia empregada nos Exemplos 2 e 6, a reconstrução inicia-se supondo um domínio com uma carga ($m = 1$). Em seguida, realiza-se esse mesmo procedimento agora supondo duas cargas no domínio ($m = 2$). As Figuras 5.27, 5.28, 5.29 e 5.30 mostram os resultados obtidos na medida em que o valor de m aumenta. Note que a localização das cargas para $m = 3$ e $m = 4$ não mudam. Além disso, para $m = 4$, o método reconstrói uma quarta carga com intensidade desprezível. Os valores das intensidades obtidas para $m = 4$ são $\alpha_1 = 8.000005$, $\alpha_2 = 7.999996$, $\alpha_3 = 8.000001$ e $\alpha_4 = -0.000006$. Assim, pode-se concluir que o número correto de cargas é $m^* = 3$.

Supondo $\lambda_2 = +1$, considera-se as mesmas localizações das cargas anteriormente utilizadas, agora com intensidades $\alpha_1^* = 3$, $\alpha_2^* = 8$ e $\alpha_2^* = 13$. A Tabela 5.11 mostra os resultados obtidos para cada valor de m. Novamente, observe que o método reconstrói três cargas de intensidades próximas para $m = 3$ e $m = 4$. Além disso, uma quarta carga com intensidade desprezível é obtida para $m = 4$, configurando o critério de parada do método. As localizações das cargas para $m = 3$ e $m = 4$ são

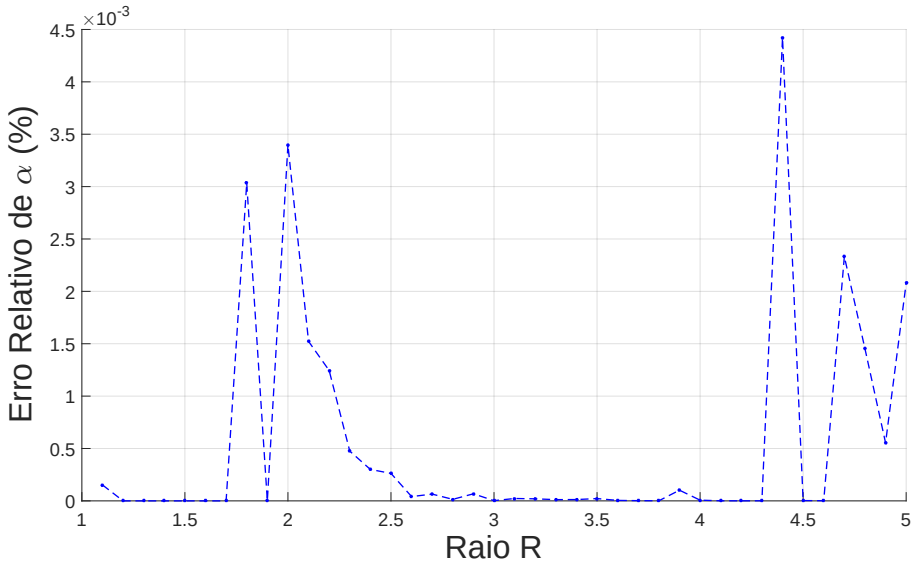

Figura 5.23: Erro relativo de α na reconstrução tridimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 2$ e $\lambda_1 = +9.5$, com $x \in X$.

reconstruídas de modo exato.

Tabela 5.11: Reconstrução tridimensional da fonte para $m = 1, 2, 3$ e 4, com $\lambda_2 = +1$.

m/α_i	α_1	α_2	α_3	α_4
$m = 1$	20.096219	$---$	$---$	$---$
$m = 2$	14.314856	9.760266	$---$	$---$
$m = 3$	12.999994	8.000010	3.000004	$---$
$m = 4$	12.865902	7.849365	2.660557	0.613036

Exemplo 11

Para analisar a reconstrução da fonte na presença de ruídos, supondo $\lambda_1 = +9.5$, considera-se uma fonte com três cargas puntuais localizadas em $x_1^* = (0; 0.27; 0.27)$, $x_2^* = (0; -0.81; -0.27)$ e $x_3^* = (-0.54; -0.54; -0.27)$, com intensidades $\alpha_1^* = 5$, $\alpha_2^* = 10$ e $\alpha_3^* = 15$, em que o fluxo q^* é poluído com o ruído gaussiano branco. Na Figura 5.31, tem-se a solução exata desejada. A Tabela 5.12 ilustra os resultados das intensidades para cada nível de ruído. Observa-se que as intensidades reconstruídas não apresentam grandes diferenças em relação aos seus valores exatos.

Fazendo esse mesmo procedimento para $\lambda_2 = +1$, considera-se as mesmas localizações das cargas anteriormente utilizadas, agora com intensidade $\alpha^* = 5$ para todas elas. A Tabela 5.13 mostra os resultados obtidos. Observe que, novamente, as intensidades reconstruídas não apresentam discrepâncias em relação aos seus valores exatos. As localizações são reconstruídas de maneira exata tanto para λ_1 quanto

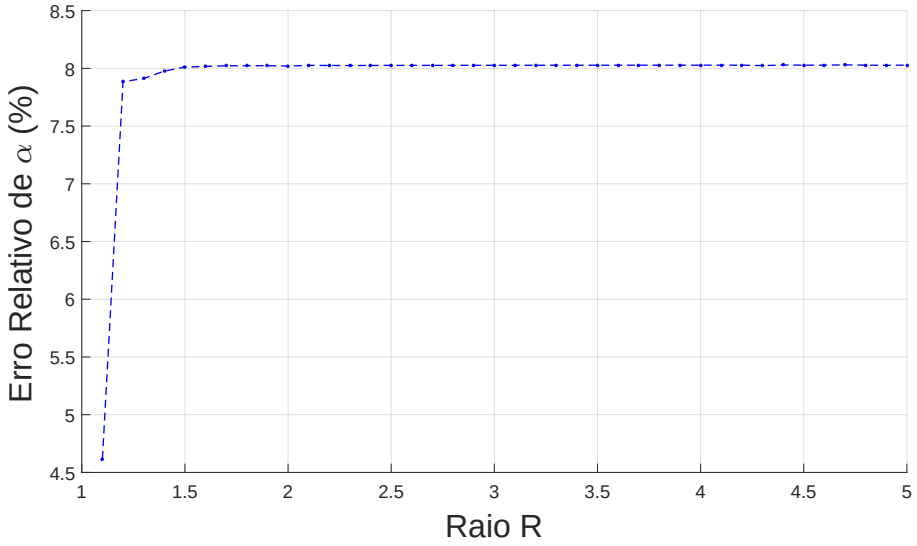

Figura 5.24: Erro relativo de α na reconstrução tridimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 2$ e $\lambda_1 = +9.5$, com $x \notin X$.

para λ_2.

Tabela 5.12: Intensidade das cargas puntuais com inserção de ruído gaussiano branco em um domínio tridimensional, considerando $\lambda_1 = +9.5$.

μ_i/α_i	α_1	α_2	α_3
$\mu = 0\%$	15.0000009	10.000001	4.999998
$\mu = 5\%$	15.2058761	9.6396458	5.2892578
$\mu = 10\%$	15.5224278	9.9212332	5.1093341
$\mu = 20\%$	15.0878255	9.8447525	5.1929335
$\mu = 40\%$	14.0299749	8.8283101	5.6872884

5.3 Equação de Helmholtz

Nesta seção, são apresentados dois experimentos para ilustrar o desempenho do método de reconstrução com relação a dados ruidosos no problema de Helmholtz, em que os resultados são obtidos considerando o domínio bidimensional e tridimensional utilizados nos exemplos anteriores.

Exemplo 12

Assim como no Exemplo 11, neste exemplo, analisa-se o comportamento do algoritmo de reconstrução em relação ao ruído nos dados de Cauchy, utilizando a equação de Helmholtz com $\lambda_3 = -4$ num domínio bidimensional. Para isso, considera-se uma fonte com três cargas puntuais de localizações $x_1^* = (0.01; -0.1)$, $x_2^* = (-0.34; 0.59)$ e $x_3^* = (0.85; 0.16)$ e intensidades $\alpha_1^* = 6$, $\alpha_2^* = 14$ e $\alpha_3^* = 17$. O número de pontos

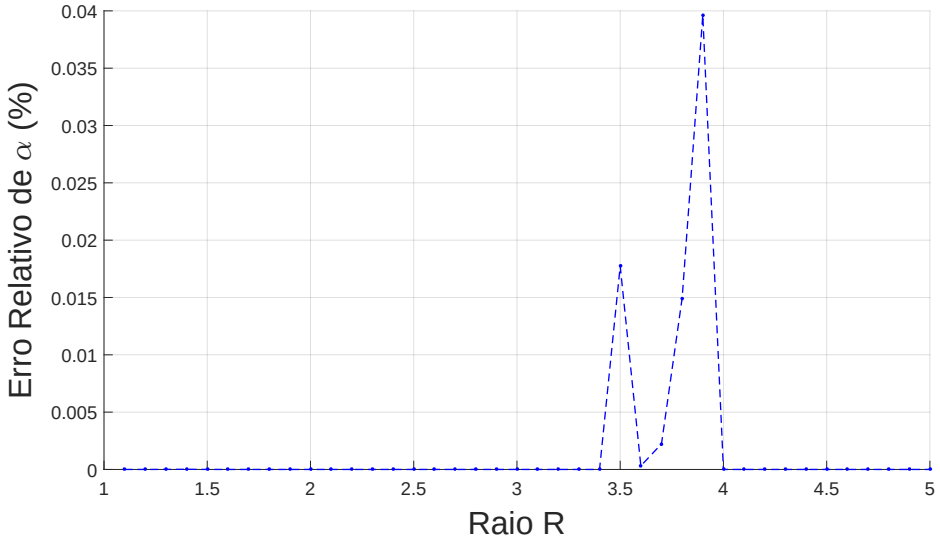

Figura 5.25: Erro relativo de α na reconstrução tridimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 4$ e $\lambda_2 = +1$, com $x \in X$.

Tabela 5.13: Intensidade das cargas puntuais com inserção de ruído gaussiano branco em um domínio tridimensional, considerando $\lambda_2 = +1$.

μ_i/α_i	α_1	α_2	α_3
$\mu = 0\%$	5.000004	4.999999	4.999995
$\mu = 5\%$	5.039607	5.036581	4.974730
$\mu = 10\%$	5.143392	5.087497	4.961763
$\mu = 20\%$	4.583630	5.669916	4.557205
$\mu = 40\%$	5.790316	4.466364	6.023838

fontes e de colocação são ajustados para $N_3 = 10$ e $L_3 = 35$, respectivamente, com raio $R = 2.4$ para a fronteira fictícia. Esse ajuste é realizado conforme explicado nos Exemplo 4 e 8. A Figura 5.32 ilustra a solução exata a ser reconstruída e a Tabela 5.14 mostra as intensidades obtidas na reconstrução para cada nível de ruído.

O método obteve êxito na reconstrução das cargas com um nível de até $\mu = 20\%$ de ruído, com localizações exatas. Entretanto, para um nível de $\mu = 40\%$ de ruído, apenas a carga 2 é reconstruída, com localização exata. A reconstrução das cargas 1 e 3 apresentam intensidades muito distantes dos seus valores exatos, com localizações incorretas.

Exemplo 13

Neste exemplo, admite-se $\lambda_3 = -4$ na reconstrução de uma fonte concentrada com três cargas puntuais para analisar o comportamento do método com relação a dados poluído com ruído na equação de Helmholtz num domínio tridimensional. As localizações das cargas são $x_1^* = (0; 0; 0)$, $x_2^* = (0.34; -0.34; 0.68)$ e $x_3^* = (0.68; -0.34; 0)$,

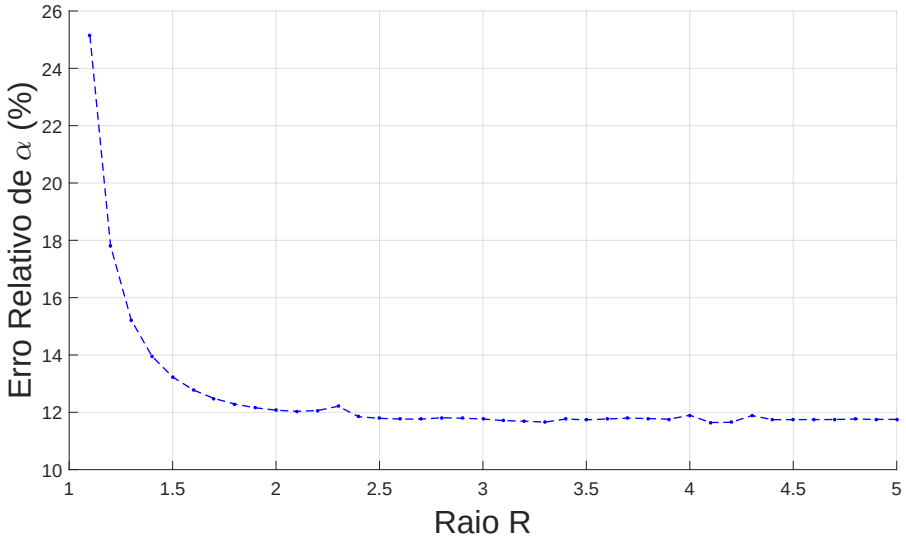

Figura 5.26: Erro relativo de α na reconstrução tridimensional de uma carga puntual com diferentes valores de R para $\alpha^* = 4$ e $\lambda_2 = +1$, com $x \notin X$.

Tabela 5.14: Intensidade das cargas com inserção de ruído gaussiano branco em um domínio tridimensional para $\lambda = -4$.

μ/α_i	α_1	α_2	α_3
$\mu = 0\%$	16.99999	13.99999	5.99999
$\mu = 5\%$	17.16730	13.67995	5.99652
$\mu = 10\%$	16.67608	13.23693	6.52708
$\mu = 20\%$	17.81924	12.10823	7.00565
$\mu = 40\%$	50.81092	13.65524	-35.13265

todas com intensidade de $\alpha^* = 9$. Calibra-se para $N_3 = L_3 = 56$ o número de pontos fonte e de colocação, com raio $R = 2$ para a fronteira fictícia. A Figura 5.33 mostra a solução exata da fonte. As localizações das cargas são reconstruídas de modo exato, com variações nas intensidades. A Tabela 5.15 mostra as intensidades reconstruídas para cada nível de ruído. Observe que não há uma discrepância significativa entre os valores aproximados e exatos, até mesmo para um nível de $\mu = 40\%$ de ruído.

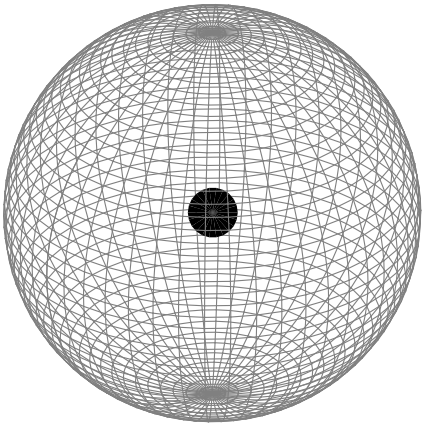

Figura 5.27: Reconstrução tridimensional da fonte para uma carga ($m = 1$)

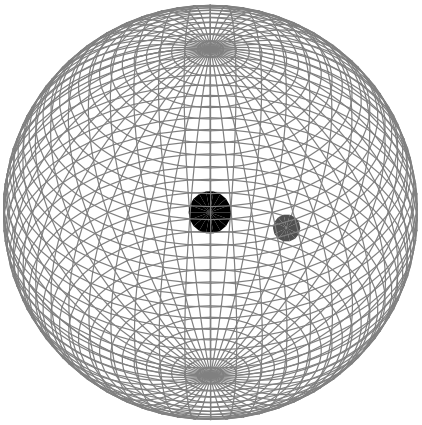

Figura 5.28: Reconstrução tridimensional da fonte para duas cargas ($m = 2$)

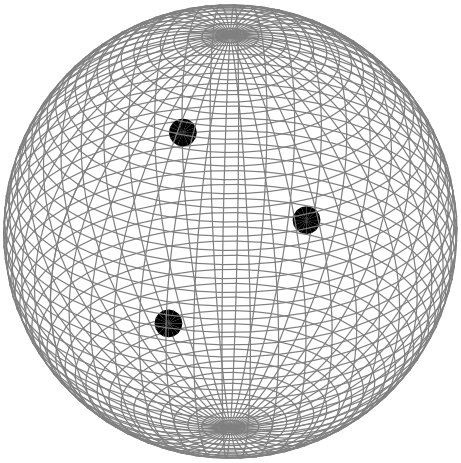

Figura 5.29: Reconstrução tridimensional da fonte para três cargas ($m = 3$)

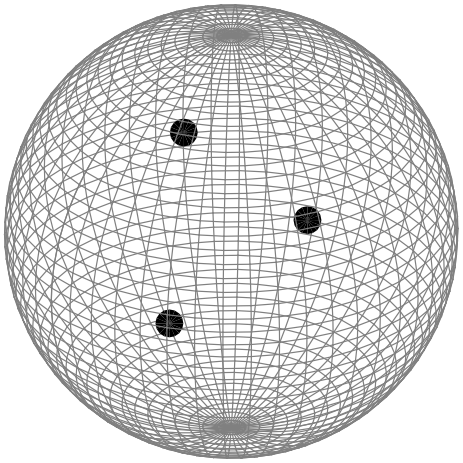

Figura 5.30: Reconstrução tridimensional da fonte para quatro cargas ($m = 4$)

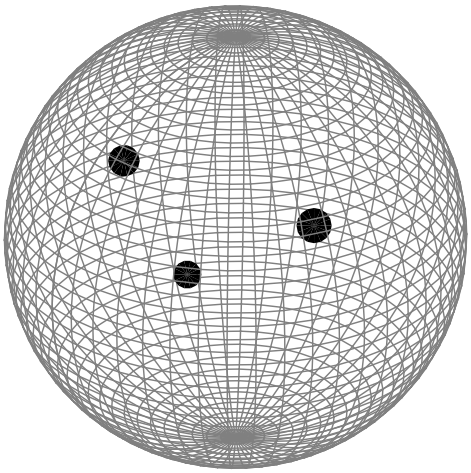

Figura 5.31: Solução tridimensional exata a ser reconstruída com dados ruidosos, com $\lambda_1 = +9.5$.

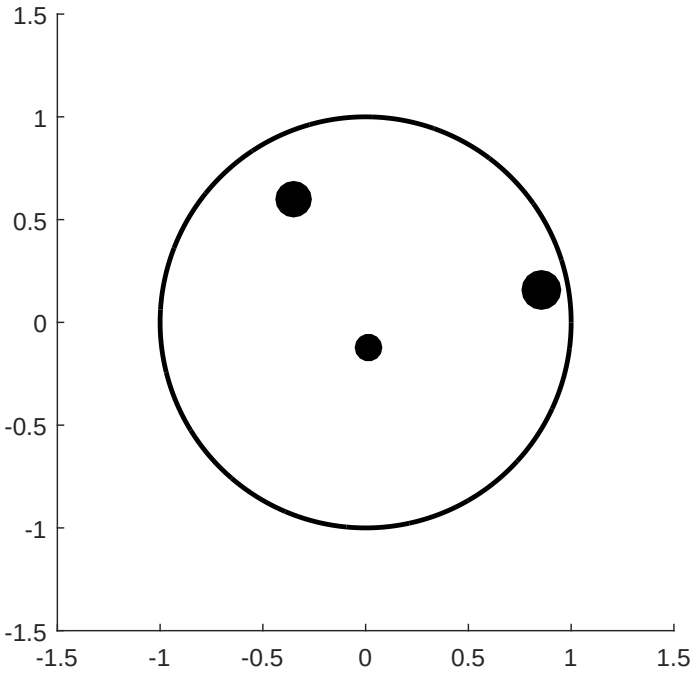

Figura 5.32: Solução bidimensional exata a ser reconstruída com dados ruidosos, com $\lambda = -4$.

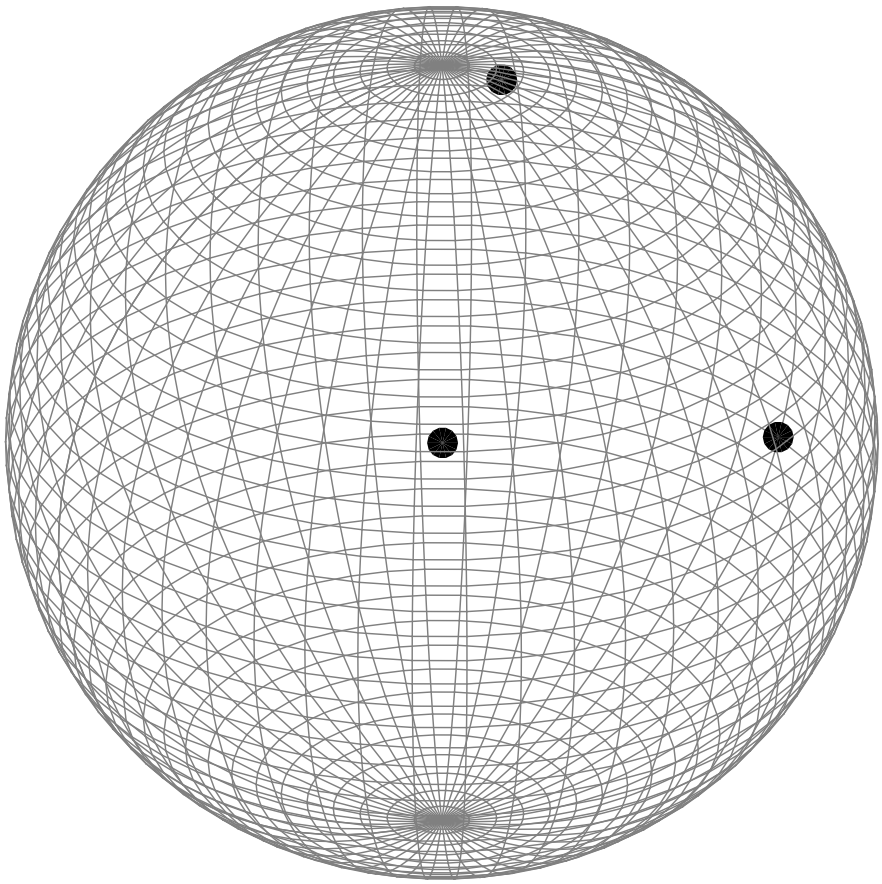

Figura 5.33: Solução tridimensional exata a ser reconstruída com dados ruidosos, com $\lambda = -4$.

Tabela 5.15: Intensidade das cargas com inserção de ruído gaussiano branco em um domínio tridimensional para $\lambda = -4$.

μ/α_i	α_1	α_2	α_3
$\mu = 0\%$	8.98954	9.00023	9.00674
$\mu = 5\%$	9.06314	9.11324	9.03730
$\mu = 10\%$	9.44144	9.38592	8.62775
$\mu = 20\%$	8.55221	9.50156	8.51140
$\mu = 40\%$	7.82378	6.69673	10.9053

Capítulo 6

Conclusões

Neste trabalho, o algoritmo proposto para a reconstrução de fontes concentradas em problemas elípticos, utilizando análise de sensibilidade e o MSF, obteve, como demonstrado através dos testes realizados, uma boa performance em relação à reconstrução de uma fonte com até três cargas puntuais consideradas em um domínio bidimensional e tridimensional.

Como é visto nos experimentos numéricos dos Exemplos 4 e 8, dado um valor para λ da equação do tipo Helmholtz e um par (u^*, q^*), é possível ajustar o número e as localizações dos pontos fonte do MSF, bem como a quantidade de pontos de colocação sobre a fronteira física, de modo a obter resultados precisos na reconstrução da fonte como é visto nos demais exemplos. Nos Exemplos 5 e 9, considerando uma quantidade pequena de elementos do conjunto de localizações admissíveis, obteve-se uma intensidade próxima do valor exato, em que as localizações são obtidas de modo exato em ambos os exemplos. Os Exemplos 6 e 10 mostram que é possível representar uma fonte concentrada com até três cargas puntuais, e gerar sinteticamente o par de dados (u^*, q^*) por meio do MSF. Além disso, o método de reconstrução identifica o número correto de cargas no interior do domínio caracterizada por uma entrada desprezível no vetor de intensidades ótimas. Nos Exemplos 3, 7, 11, 12 e 13, são realizados testes com inserção de ruído no fluxo gerado pelo MSF. Mesmo com um nível de até 20% de ruído, a aproximação das cargas fica próxima de seus valores exatos.

O uso do MSF permite representar as cargas puntuais com inserção de pontos fonte no interior do domínio, facilitando a etapa de validação do algoritmo de reconstrução proposto. Vale salientar que a matriz H e o vetor d usados no cálculo da variação do funcional são calculados utilizando a integração numérica sobre a fronteira do domínio, resultando em um melhor custo computacional para a obtenção da solução do problema inverso. Assim, conclui-se que o método proposto para a solução do problema inverso de reconstrução de fontes concentradas é eficaz, robusto e eficiente, levando em conta os resultados obtidos para uma quantidade pequena

de pontos do *grid*. Como trabalhos futuros, sugere-se a reconstrução do termo fonte estudado considerando leituras parciais sobre a fronteira do domínio e, além disso, um estudo acerca do ajuste do número de pontos fonte e de colocação para um dado valor não nulo de λ, com o intuito de obter uma reconstrução mais precisa da fonte.

Referências Bibliográficas

[1] CJS Alves and CS Chen. A new method of fundamental solutions applied to nonhomogeneous elliptic problems. *Advances in Computational Mathematics*, 23(1-2):125–142, 2005.

[2] SR Arridge. Optical tomography in medical imaging. *Inverse problems*, 15(2):R41, 1999.

[3] F Barthelmes and R Dietrich. Use of point masses on optimized positions for the approximation of the gravity field. In *Determination of the Geoid*, pages 484–493. Springer, 1991.

[4] J Baumeister and A Leitao. Topics in inverse problems, 25, brazilian mathematics coll, 2005.

[5] M Bertero and P Boccacci. *Introduction to inverse problems in imaging*. CRC press, 1998.

[6] H Bertete-Aguirre, E Cherkaev, and M Oristaglio. Non-smooth gravity problem with total variation penalization functional. *Geophysical Journal International*, 149(2):499–507, 2002.

[7] S Bilicz, M Lambert, et al. Solution of inverse problems in nondestructive testing by a kriging-based surrogate model. *IEEE Transactions on Magnetics*, 48(2):495–498, 2012.

[8] H Brezis. *Functional analysis, Sobolev spaces and partial differential equations*. Springer Science & Business Media, 2010.

[9] J Burkardt. *Grid Points Within a 3D Ball*, 2010 (acessado em 3 de setembro, 2018). `https://people.sc.fsu.edu/~jburkardt/m_src/ball_grid/ball_grid.html`.

[10] J Burkardt. *Grids on a Sphere*, 2012 (acessado em 18 de setembro, 2018). `https://people.sc.fsu.edu/~jburkardt/m_src/sphere_grid/sphere_grid.html`.

[11] E Casas and E Zuazua. Spike controls for elliptic and parabolic pdes. *Systems & Control Letters*, 62(4):311–318, 2013.

[12] W Chen, Zhuo-Jia Fu, and Ching-Shyang Chen. *Recent advances in radial basis function collocation methods*. Springer, 2014.

[13] RS Crosson. Crustal structure modeling of earthquake data: 1. simultaneous least squares estimation of hypocenter and velocity parameters. *Journal of geophysical research*, 81(17):3036–3046, 1976.

[14] RMG da Silva. *Problema Inverso de Reconstrução e Identificação de Fontes em Equações Elípticas*. PhD thesis, Universidade Federal do Rio de Janeiro, 2016.

[15] A El Badia and T Ha-Duong. An inverse source problem in potential analysis. *Inverse Problems*, 16(3):651, 2000.

[16] A El Badia and T Ha-Duong. On an inverse source problem for the heat equation. application to a pollution detection problem. *Journal of inverse and ill-posed problems*, 10(6):585–599, 2002.

[17] A El Badia, T Ha-Duong, and A Hamdi. Identification of a point source in a linear advection–dispersion–reaction equation: application to a pollution source problem. *Inverse Problems*, 21(3):1121, 2005.

[18] A El Badia and T Nara. An inverse source problem for helmholtz's equation from the Cauchy data with a single wave number. *Inverse Problems*, 27(10):105001, 2011.

[19] G Fairweather and A Karageorghis. The method of fundamental solutions for elliptic boundary value problems. *Advances in Computational Mathematics*, 9(1-2):69, 1998.

[20] G Fairweather, A Karageorghis, and PA Martin. The method of fundamental solutions for scattering and radiation problems. *Engineering Analysis with Boundary Elements*, 27(7):759–769, 2003.

[21] O Faugeras, G Clément, et al. *The inverse EEG and MEG problems: The adjoint state approach I: The continuous case*. PhD thesis, INRIA, 1999.

[22] MA Golberg and CS Chen. The method of fundamental solutions for potential, helmholtz and diffusion problems. *Boundary integral methods: numerical and mathematical aspects*, 1:103–176, 1998.

[23] AV Goncharsky and SY Romanov. Inverse problems of ultrasound tomography in models with attenuation. *Physics in Medicine & Biology*, 59(8):1979, 2014.

[24] AV Goncharsky, SY Romanov, and SY Seryozhnikov. Inverse problems of 3d ultrasonic tomography with complete and incomplete range data. *Wave motion*, 51(3):389–404, 2014.

[25] CW Groetsch. The theory of tikhonov regularization for fredholm equations. *104p, Boston Pitman Publication*, 1984.

[26] RM Gulrajani. The forward and inverse problems of electrocardiography. *IEEE Engineering in Medicine and Biology Magazine*, 17(5):84–101, 1998.

[27] J Hadamard and PM Morse. Lectures on Cauchy's problem in linear partial differential equations. *Physics Today*, 6:18, 1953.

[28] S He and VG Romanov. Identification of dipole sources in a bounded domain for maxwell's equations. *Wave motion*, 28(1):25–40, 1998.

[29] BKP Horn and MJ Brooks. The variational approach to shape from shading. *Computer Vision, Graphics, and Image Processing*, 33(2):174–208, 1986.

[30] V Isakov, S Leung, and J Qian. A fast local level set method for inverse gravimetry. *Communications in Computational Physics*, 10(4):1044–1070, 2011.

[31] J Kalifa, S Mallat, et al. Thresholding estimators for linear inverse problems and deconvolutions. *The Annals of Statistics*, 31(1):58–109, 2003.

[32] AD Klose, V Ntziachristos, and AH Hielscher. The inverse source problem based on the radiative transfer equation in optical molecular imaging. *Journal of Computational Physics*, 202(1):323–345, 2005.

[33] RV Kohn and M Vogelius. Relaxation of a variational method for impedance computed tomography. *Communications on Pure and Applied Mathematics*, 40(6):745–777, 1987.

[34] V Komornik and M Yamamoto. Upper and lower estimates in determining point sources in a wave equation. *Inverse Problems*, 18(2):319, 2002.

[35] S Kubo. Inverse problems related to the mechanics and fracture of solids and structures. *JSME international journal. Ser. 1, Solid mechanics, strength of materials*, 31(2):157–166, 1988.

[36] VI Lebedev and DN Laikov. A quadrature formula for the sphere of the 131st algebraic order of accuracy. In *Doklady Mathematics*, volume 59, pages 477–481. Pleiades Publishing, Ltd., 1999.

[37] TJ Machado. *Um novo método para reconstrução de fontes concentradas*. PhD thesis, Laboratório Nacional de Computação Científica - LNCC, 2017.

[38] RS MacLeod and DH Brooks. Recent progress in inverse problems in electrocardiology. *IEEE Engineering in Medicine and Biology Magazine*, 17(1):73–83, 1998.

[39] BA Mair. Tikhonov regularization for finitely and infinitely smoothing operators. *SIAM Journal on Mathematical Analysis*, 25(1):135–147, 1994.

[40] VA Morozov and M Stessin. *Regularization methods for ill-posed problems*. CRC press Boca Raton, FL:, 1993.

[41] M Nair, M Hegland, and R Anderssen. The trade-off between regularity and stability in tikhonov regularization. *Mathematics of Computation of the American Mathematical Society*, 66(217):193–206, 1997.

[42] F Natterer. *The mathematics of computerized tomography*, volume 32. Siam, 1986.

[43] F Natterer and F Wübbeling. *Mathematical methods in image reconstruction*, volume 5. Siam, 2001.

[44] AA Novotny and TJ Machado. Um novo método de reconstrução de fontes concentradas. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, 3(1), 2015.

[45] HRB Orlande, GS Dulikravich, and MJ Colaço. Application of bayesian filters to heat conduction problem. *EngOpt*, pages 1–5, 2008.

[46] R Parrish. *getLebedevSphere*, 2010 (acessado em 3 de setembro, 2018). `https://www.mathworks.com/matlabcentral/fileexchange/27097-getlebedevsphere`.

[47] C Prada, E Kerbrat, et al. Time reversal techniques in ultrasonic nondestructive testing of scattering media. *Inverse problems*, 18(6):1761, 2002.

[48] T Rudy and BJ Messinger-Rapport. The inverse problem in electrocardiography: solutions in terms of epicardial potentials. *Critical reviews in biomedical engineering*, 16(3):215–268, 1988.

[49] AN Tikhonov and VY Arsenin. Solutions of ill-posed problems, vh winston and sons, washington, dc, 1977. *Translated from Russian*.

[50] H Vogel. A better way to construct the sunflower head. *Mathematical biosciences*, 44(3-4):179–189, 1979.

[51] KA Woodbury. What are inverse problems. *disponível em http://www.me.ua.edu/inverse/whatis.html*, 1995.

[52] MS Zhdanov. *Geophysical inverse theory and regularization problems*, volume 36. Elsevier, 2002.

Apêndice A

Códigos do Programa

Equação de Poisson

Programa Principal

```
%Ex. 1
function [valor_alfa, Pts_otimo] = mainLap_2D(alpha, xyst,L_Grid)

%Ex. 2
% function [valor_alfa, Pts_otimo, xyc, xys, xygr] = ...
    mainLap_2D(alpha, xyst,L_Grid,m)

%Ex. 3
% function [valor_alfa, Pts_otimo, q_star_without_noise, ...
    q_star_with_noise, xyc, xys, xygr] = mainLap_2D(alpha, xyst, ...
    noise, L_Grid)

% Ex. Ajuste (ajuste de parametros para o Ex 1)
%  function [valor_alfa, Pts_otimo, xyc, xys, xygr] = ...
    mainLap_2D(alpha, xyst, L_Grid,R)
%%
%Tolerancia para a funcao lsqr
tol=1e-15;

%Raio do dominio
r = 1;
%Raio da fronteira ficticia
R = 4.0;
%Numero de pontos de colocacao para solucao homogenea
M = 30;
%numero de pontos de fonte para solucao homogenea
N = 15;
% Pontos de fonte e de colocacao
```

```
25 xc = zeros(M,1);  yc = xc;
26 xs = zeros(N,1);  ys = xs;
27
28 %vetor normal
29 nxe = xc;  nye = xc;
30
31 for k = 1:M
32    nxe(k,1) = cos(2*pi*k/M);
33    nye(k,1) = sin(2*pi*k/M);
34 end
35 xc = r*nxe;
36 yc = r*nye;
37 nxye = [nxe nye];
38 u_star = nxe;
39
40 for k = 1:N
41    xs(k,1) = R*cos(2*pi*k/N);
42    ys(k,1) = R*sin(2*pi*k/N);
43 end
44
45 xyc = [ xc yc ];
46 xys = [ xs ys ];
47
48 %Numero de pontos de fonte e de colocacao para solucao ...
      particular: Mp, Np
49 [xcp ycp, Mp] = calcPontIntCirc(15,0.05,r-0.1);
50 [xsp ysp, Np] = calcPontIntCirc(8,0.07,r-0.1);
51
52 %Numero de cargas para serem reconstruidas
53 numPtsfontcurr = size(alpha,1);
54
55 %numero de pontos para integracao de linha
56 MT = 25;
57
58 %pontos para integracao numerica
59 x_int = zeros(MT,1);  y_int = x_int;
60
61 %Vetor normal
62 nxe_int = x_int;  nye_int = x_int;
63
64 for k = 1:MT
65    nxe_int(k,1) = cos(2*pi*k/MT);
66    nye_int(k,1) = sin(2*pi*k/MT);
67 end
68 nxye_int = [nxe_int nye_int];
69 x_int = r*nxe_int;
70 y_int = r*nye_int;
```

```
71 xy_int = [x_int y_int];
72
73 u_star_integral = nxe_int;
74
75 % Dados de Cauchy
76 q_star = calc_q_star_lap_2D(xyc,xys,alpha,...
77     xyst,u_star,nxye,numPtsfontcurr,tol);
78 q_star_integral = calc_q_star_lap_2D(xy_int,xys,...
79     alpha,xyst,u_star_integral,nxye_int,numPtsfontcurr,tol);
80
81 % %acrescenta ruido
82 % s = rng;
83 % q_star_without_noise = q_star;
84 % q_star_with_noise = q_star.*(1+ noise*(2*rand(M,1)-1)/100 );
85 %
86 % rng(s);
87 % q_star_integral_without_noise = q_star_integral;
88 % q_star_integral_with_noise = q_star_integral.*(1+ ...
      noise*(2*randn(MT,1)-1)/100);
89 %
90 % q_star = q_star_with_noise;
91 % q_star_integral = q_star_integral_with_noise;
92
93 areaDom = pi*r^2;
94 %constante de compatibilidade
95 cteComp = T_Simpson_2D(q_star_integral, (2*pi*r)/MT)/areaDom;
96
97 % funcao constante do problema h_i
98 ctehi = 1/areaDom;
99
100 % Numero de onda (Vide CJS Alves and CS Chen)
101 q = 8;
102 vetLamb = -[1 4 9 16 25 36 49 64 81 100];
103 %%
104 %Calcula os coeficientes de uN e hi particular
105 raio=[];
106 M_uNP_hi_aux=[];
107 M_uNP_hi=[];
108 for k=1:q
109     for i=1:Mp
110         for j=1:Np
111             raio2 = (xcp(i) - xsp(j))^2 + (ycp(i) - ysp(j))^2;
112             arg=sqrt(-vetLamb(k))*sqrt(raio2);
113             H0=besselj(0,arg);%;+ 1i*bessely(0,arg);
114             M_uNP_hi_aux(i,j) = (1i*H0)/4;
115         end
116     end
```

```
117     M_uNP_hi = [M_uNP_hi M_uNP_hi_aux];
118 end
119 vetor_uNP = -cteComp*ones(Mp,1);
120 vetor_hiP = -ctehi*ones(Mp,1);
121 
122 %coeficientes das funcoes constantes
123 c_uNP_aux = lsqr(M_uNP_hi,vetor_uNP,tol);
124 c_hiP_aux = lsqr(M_uNP_hi,vetor_hiP,tol);
125 
126 %coeficientes de uN e hi particular (uNP hiP)
127 aux5 = 1;
128 aux6 = 1;
129 for i = 1:Np*q
130     c_uNP(i,1) = c_uNP_aux(i,1)/vetLamb(aux5);
131     c_hiP(i,1) = c_hiP_aux(i,1)/vetLamb(aux5);
132     if (i==aux6*Np)
133         aux5=aux5+1;
134         aux6=aux6+1;
135     end
136 end
137 %%
138 %Calcula uN e hi particular na fronteira para integracao numerica
139 raio=[];
140 M_uNP_hi_aux=[];
141 M_uNP_hi=[];
142 for k=1:q
143     for i=1:MT
144         for j=1:Np
145             raio2 = (x_int(i) - xsp(j))^2 + (y_int(i) - ysp(j))^2;
146             arg=sqrt(-vetLamb(k))*sqrt(raio2);
147             H0=besselj(0,arg);%+ 1i*bessely(0,arg);
148             M_uNP_hi_aux(i,j) = (1i*H0)/4;
149         end
150     end
151     M_uNP_hi = [M_uNP_hi M_uNP_hi_aux];
152 end
153 
154 uNP = M_uNP_hi*c_uNP;
155 hiP = M_uNP_hi*c_hiP;
156 %%
157 %Calcula a derivada normal de uN e hi particular (DuNP DhiP) na ...
       fronteira
158 %fisica
159 raio=[];
160 M_uNP_hi_aux=[];
161 M_uNP_hi=[];
162 for k=1:q
```

```
163     for i=1:M
164         for j=1:Np
165             raio2 = (xc(i) - xsp(j))^2 + (yc(i) - ysp(j))^2 ;
166             arg=sqrt(-vetLamb(k))*sqrt(raio2);
167             H1=- besselj(1,arg);%- 1i*bessely(1,arg);
168             M_uNP_hi_aux_x(i,j) = ( 1i*H1*sqrt(-vetLamb(k))*( ...
                    (xc(i) - xsp(j)) ) )/(4*sqrt(raio2));
169             M_uNP_hi_aux_y(i,j) = ( 1i*H1*sqrt(-vetLamb(k))*( ...
                    (yc(i) - xsp(j)) ) )/(4*sqrt(raio2));
170             M_uNP_hi_aux(i,j) = M_uNP_hi_aux_x(i,j)*nxe(i) + ...
                   M_uNP_hi_aux_y(i,j)*nye(i);
171         end
172     end
173     M_uNP_hi = [M_uNP_hi M_uNP_hi_aux];
174 end
175 Deriv_front_uNP = M_uNP_hi*c_uNP;
176 Deriv_front_hiP = M_uNP_hi*c_hiP;
177 %%
178 %calcula os coeficientes de uN Homogeneo
179 M_uNH=[];
180 for i=1:M
181     for j=1:N
182         raio2 = (xc(i)-xs(j))^2 + (yc(i)-ys(j))^2;
183         %derivada normal
184         DuNH_x(i,j) = -(xc(i)-xs(j))/(2*pi*raio2);
185         DuNH_y(i,j) = -(yc(i)-ys(j))/(2*pi*raio2);
186         M_uNH(i,j)  = DuNH_x(i,j)*nxe(i) + DuNH_y(i,j)*nye(i);
187     end
188     vetor_uNH(i,1) = - ( q_star(i) + Deriv_front_uNP(i) );
189 end
190
191 %condicao da integral para a matriz
192 cond_integral_matriz = [];
193 for j=1:N
194     vet_raio  = sqrt( (x_int - xs(j)).^2 + (y_int - ys(j)).^2 );
195     vet_phi_i = -log(vet_raio)/(2*pi);
196     %"cond_integral_matriz" sera reaproveitada para calcular
197     %uN Homogeno e hi no grid
198     cond_integral_matriz(1,j) = T_Simpson_2D(vet_phi_i, 2*pi*r/MT);
199     M_uNH(M+1,j) = cond_integral_matriz(1,j);
200 end
201 %condicao da integral para o vetor
202 %uD pode ser substituido por u*
203 vetor_uNH(M+1,1) = T_Simpson_2D(u_star_integral, 2*pi*r/MT) - ...
       T_Simpson_2D(uNP, 2*pi*r/MT);
204
205 %coeficientes de uNH
```

```
206 c_uNH = lsqr(M_uNH,vetor_uNH,tol);
207 %%
208 %calcula uN nos pontos interiores auxiliando na integracao numerica
209 matriz_uNH = [];
210 for i=1:MT
211     for j=1:N
212         raio2 = (x_int(i)-xs(j))^2 + (y_int(i)-ys(j))^2;
213         matriz_uNH(i,j) = -log(sqrt(raio2))/(2*pi);
214     end
215 end
216
217 uNH = matriz_uNH*c_uNH;
218 uN = uNH + uNP;
219
220 %condicao da integral para os termos independentes no sistema
221 %em hi (sera fixo em todo o grid)
222 integral_hiP = T_Simpson_2D(hiP,2*pi*r/MT);
223
224 %%
225 tam = L_Grid;
226 [xgr ygr] = sunflower_2D(tam,0);
227 xygr = [ xgr ygr ];
228
229 [hi di] = calc_hi_di_lap_2D(...
230                     xc,yc,nxe,nye,xs,ys,x_int,y_int,...
231                     xygr,hiP,u_star_integral,uN,...
232                     Deriv_front_hiP,tol,integral_hiP,...
233                     cond_integral_matriz,tam);
234 aux = 1;
235 % numPtsfontcurr = m;
236
237 switch numPtsfontcurr
238     case 1
239         for i1 = 1:tam
240             entry_hi = hi(:,i1);
241             entry_di = di(:,i1);
242             [H d] = calc_H_d_lap_2D(entry_hi, ...
                        entry_di,numPtsfontcurr,MT,r);
243
244             alfas(:,aux) = lsqr(H,d,tol);
245             funcional(aux,1)=-0.5*alfas(:,aux)'*d;
246             aux_index(:,aux) = i1;
247             aux = aux + 1;
248         end
249     case 2
250         for i1 = 1:tam
251             for i2 = i1:tam
```

```
252                 entry_hi = [ hi(:,i2) hi(:,i1) ];
253                 entry_di = [ di(:,i2) di(:,i1) ];
254                 [H d] = calc_H_d_lap_2D(entry_hi, ...
                        entry_di,numPtsfontcurr,MT,r);
255
256                 alfas(:,aux) = lsqr(H,d,tol);
257                 funcional(aux,1)=-0.5*alfas(:,aux)'*d;
258                 aux_index(:,aux) = [i2 i1];
259                 aux = aux + 1;
260             end
261         end
262     case 3
263         for i1 = 1:tam
264             for i2 = i1:tam
265                 for i3 = i2:tam
266                     entry_hi = [ hi(:,i3) hi(:,i1) hi(:,i2)];
267                     entry_di = [ di(:,i3) di(:,i1) di(:,i2)];
268                     [H d] = calc_H_d_lap_2D(entry_hi, ...
                            entry_di,numPtsfontcurr,MT,r);
269
270                     alfas(:,aux) = lsqr(H,d,tol);
271                     funcional(aux,1)=-0.5*alfas(:,aux)'*d;
272                     aux_index(:,aux) = [i3 i1 i2];
273                     aux = aux + 1;
274                 end
275             end
276         end
277     case 4
278         for i1 = 1:tam
279             for i2 = i1:tam
280                 for i3 = i2:tam
281                     for i4 = i3:tam
282                         entry_hi = [ hi(:,i2) hi(:,i3) hi(:,i4) ...
                                hi(:,i1)];
283                         entry_di = [ di(:,i2) di(:,i3) di(:,i4) ...
                                di(:,i1)];
284                         [H d] = calc_H_d_lap_2D(entry_hi, ...
                                entry_di,numPtsfontcurr,MT,r);
285
286                         alfas(:,aux) = lsqr(H,d,tol);
287                         funcional(aux,1)=-0.5*alfas(:,aux)'*d;
288                         aux_index(:,aux) = [i2 i3 i4 i1];
289                         aux = aux + 1;
290                     end
291
292                 end
293             end
```

```
294         end
295
296     otherwise
297 end
298
299 [value, index] = min(funcional);
300 funcional  = value;
301 valor_alfa = alfas(:,index);
302 Pts_otimo = xygr(aux_index(:,index),:);
```

Rotinas Utilizadas

```
1  %Calcula pontos no interior do circulo
2  function [x, y, tam]=calcPontIntCirc(L,dr,R)
3  tet=0:pi/(L/2):2*pi;
4  r=0:dr:R;
5  aux=1;
6  x=zeros((L-1)*length(r),1);
7  y=x;
8  for i=1:length(r)
9      for k = 1:L
10         x(aux,1) = sqrt(r(i))*cos(tet(k));
11         y(aux,1) = sqrt(r(i))*sin(tet(k));
12         aux=aux+1;
13     end
14 end
15 tam = aux-1;
```

```
1  function q_star=calc_q_star_lap_2D(xyc,xys,...
2      alpha,xyst,u_star,nxye,numPtsfontcurr,tol)
3  %Numero de pontos de colocacao
4  [l1 c1] = size(xyc);
5  M = l1;
6  %Numero de pontos de fonte
7  [l2 c2] = size(xys);
8  N = l2;
9
10 %Calcula a derivada normal e a matriz do sistema
11 for i = 1:M
12     %Parcela da solucao aproximada associada ao somatorio
13     for j = 1:N
14         M_qst_H(i,j) = (xyc(i,1) - xys(j,1))^2 + (xyc(i,2) - ...
               xys(j,2))^2;
15         Gx_H(i,j) =  -(xyc(i,1) - xys(j,1))/(2*pi*(M_qst_H(i,j)));
16         Gy_H(i,j) =  -(xyc(i,2) - xys(j,2))/(2*pi*(M_qst_H(i,j)));
```

```
17          AN(i,j) = Gx_H(i,j)*nxye(i,1) + Gy_H(i,j)*nxye(i,2);
18      end
19      fun(i,1)= u_star(i);
20      q_star2(i,1) = 0;
21      for k=1:numPtsfontcurr
22          M_qst(i,1,k) = (xyc(i,1) - xyst(k,1))^2 + (xyc(i,2) - ...
                xyst(k,2))^2;
23          fun(i,1) = fun(i,1) + ...
                alpha(k,1)*log(sqrt(M_qst(i,1,k)))/(2*pi);
24          %parcela da solucao aprox associada ao alpha1 e alpha2
25          Gx(i,1,k) =  -(xyc(i,1) - xyst(k,1))/(2*pi*(M_qst(i,1,k)));
26          Gy(i,1,k) =  -(xyc(i,2) - xyst(k,2))/(2*pi*(M_qst(i,1,k)));
27          q_star2(i,1) = q_star2(i,1) +...
28              alpha(k,1)*( Gx(i,1,k)*nxye(i,1) + ...
                    Gy(i,1,k)*nxye(i,2) );
29      end
30  end
31  AD = - log(sqrt(M_qst_H))/(2*pi);
32  cD = lsqr(AD,fun,tol);
33  q_star1=AN*cD;
34  q_star = -( q_star1 + q_star2 );
```

```
1   function [hi di] = calc_hi_di_lap_2D(...
2                               xc,yc,nxe,nye,xs,ys,x_int,y_int,...
3                               pts_gr,hiP,u_star_integral,uN,...
4                               Deriv_front_hiP,tol,integral_hiP,...
5                               cond_integral_matriz,numPtsfontcurr)
6   %%
7   %Numero de pontos de colocacao
8   M = size(xc,1);
9   %Numero de pontos fonte
10  N = size(xs,1);
11  %Numero de pontos para integracao numerica
12  MT = size(x_int,1);
13
14  %Calcula a derivada normal de vi e os coeficientes de hi
15  matriz_vi = [];
16  for i=1:M
17      for j=1:N
18          raio2 = (xc(i) - xs(j))^2 + (yc(i) - ys(j))^2;
19          matriz_vi(i,j) = -log(sqrt(raio2))/(2*pi);
20
21          matriz_Dvi1_hiH_x(i,j) = -( xc(i) - xs(j) )/(2*pi*raio2);
22          matriz_Dvi1_hiH_y(i,j) = -( yc(i) - ys(j) )/(2*pi*raio2);
23          matriz_Dvi1_hiH(i,j)   = matriz_Dvi1_hiH_x(i,j)*nxe(i) + ...
                matriz_Dvi1_hiH_y(i,j)*nye(i);
```

```
24      end
25      for k = 1:numPtsfontcurr
26          dist_vet(i,1,k) = (xc(i) - pts_gr(k,1))^2 + (yc(i) - ...
                pts_gr(k,2))^2;
27          vetor_vi(i,1,k) = log(sqrt(dist_vet(i,1,k)))/(2*pi);
28
29          Dvi2_x(i,1,k) =  -( xc(i) - pts_gr(k,1) ...
                )/(2*pi*dist_vet(i,1,k));
30          Dvi2_y(i,1,k) =  -( yc(i) - pts_gr(k,2) ...
                )/(2*pi*dist_vet(i,1,k));
31          Dvi2(i,1,k)   = Dvi2_x(i,1,k)*nxe(i) + Dvi2_y(i,1,k)*nye(i);
32      end
33  end
34
35  for k=1:numPtsfontcurr
36      cvi(:,k) = lsqr(matriz_vi,vetor_vi(:,1,k),tol);
37      Dvi1(:,k) = matriz_Dvi1_hiH*cvi(:,k);
38      Dvi(:,k) = Dvi1(:,k) + Dvi2(:,1,k);
39      vetor_hiH(:,1,k) = - (Dvi(:,k) + Deriv_front_hiP);
40  end
41
42  %Condicao da integral para a matriz do sistema
43  matriz_Dvi1_hiH(M+1,:) = cond_integral_matriz(1,:);
44  %condicao da integral para os termos independentes
45  for k = 1:numPtsfontcurr
46      vetor_hiH(M+1,1,k) = - integral_hiP;
47  end
48
49  %Coeficientes de hi homogeneo
50  for k=1:numPtsfontcurr
51      chiH(:,k) = lsqr(matriz_Dvi1_hiH,vetor_hiH(:,1,k),tol);
52  end
53  %%
54  %calcular matriz H e vetor d
55  for i=1:MT
56      for j=1:N
57          raio(i,j) = sqrt( (x_int(i) - xs(j))^2 + (y_int(i) - ...
                ys(j))^2 );
58          matriz_hiH(i,j) = -log(raio(i,j))/(2*pi);
59      end
60      for k=1:numPtsfontcurr
61          hi(i,k) = matriz_hiH(i,:)*chiH(:,k) + hiP(i);
62          di(i,k) = hi(i,k)*(u_star_integral(i) - uN(i));
63      end
64  end
```

```
function [H d] = calc_H_d_lap_2D(hi, di,numPtsfontcurr,MT,r)
for i = 1:numPtsfontcurr
    for j=i:numPtsfontcurr
        if (i==j)
            funHij = hi(:,i).^2;
            H(i,j) = T_Simpson_2D(funHij, 2*pi*r/MT);
        else
            funHij = hi(:,i).*hi(:,j);
            H(i,j) = T_Simpson_2D(funHij, 2*pi*r/MT);
            H(j,i) = H(i,j);
        end
    end
    d(i,1) = T_Simpson_2D(di(:,i), 2*pi*r/MT);
end
```

Exemplos Numéricos

```
%EXEMPLO 1
clear all
close all
clc

tic
%Intensidade das cargas
alpha(1,1) = 5;

%Localizacoes das cargas
xyst(1,1) =  0.4069;
xyst(1,2) =  0.4486;
%Tamanho do grid
L_Grid = 500;
[alfa_aprox, Pts_aprox] = mainLap_2D(alpha, xyst,L_Grid);
alpha
alfa_aprox
Pts_aprox
erro_loc = norm(xyst - Pts_aprox)
erro_int = abs(alpha - alfa_aprox)/abs(alpha)
```

```
%EXEMPLO 2
clear all
close all
clc

tic
```

```
%Intensidade das cargas
alpha(1,1) = 5;
alpha(2,1) = 10;
alpha(3,1) = 15;
%Tamanho do grid
tam = 100   ;

[p1 p2] = sunflower_2D(tam,0);
p = [p1 p2]

pto1=37;
pto2=56;
pto3=33;

%Localizacao das cargas
xyst(1,1) =   p(pto1,1);
xyst(1,2) =   p(pto1,2);

xyst(2,1) =   p(pto2,1);
xyst(2,2) =   p(pto2,2);

xyst(3,1) =   p(pto3,1);
xyst(3,2) =   p(pto3,2);

%Numero de cargas a serem reconstruidas
m = 1;
fator = 30;
[x,y,z] = cylinder(1,1000);
[alfa_aprox, Pts_aprox, xyc, xys, xygr] = mainLap_2D(alpha, ...
    xyst, tam,m);
r1 = alpha(1,1)/2;
r2 = alpha(2,1)/2;
r3 = alpha(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);

figure(1)
hold on
p1 = plot(xyst(1,1),xyst(1,2),'K.','MarkerSize', ...
    value(1)+fator,'LineWidth',2);
p2 = plot(xyst(2,1),xyst(2,2),'K.','MarkerSize', ...
    value(2)+fator,'LineWidth',2);
p3 = plot(xyst(3,1),xyst(3,2),'K.','MarkerSize', ...
    value(3)+fator,'LineWidth',2);
p4 = plot(x(1,:),y(1,:),'K','LineWidth',2)
```

```
50 hold off
51 axis([-1.5 1.5 -1.5 1.5])
52 daspect([1 1 1])
53
54 if(m==1)
55     r1 = alfa_aprox(1,1)/2;
56     value(1) = pi*(r1);
57
58     figure(2)
59     hold on
60     p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K. ','MarkerSize', ...
           value(1)+fator,'LineWidth',2);
61     p2 = plot(x(1,:),y(1,:),'K','LineWidth',2)
62     hold off
63     axis([-1.5 1.5 -1.5 1.5])
64     daspect([1 1 1])
65     Pts_aprox
66     alfa_aprox
67 end
68 if(m==2)
69     r1 = alfa_aprox(1,1)/2;
70     r2 = alfa_aprox(2,1)/2;
71
72     value(1) = pi*(r1);
73     value(2) = pi*(r2);
74
75     figure(2)
76     hold on
77     p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K.','MarkerSize', ...
           value(1)+fator,'LineWidth',2);
78     p2 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'K.','MarkerSize', ...
           value(2)+fator,'LineWidth',2);
79
80     p3 = plot(x(1,:),y(1,:),'K','LineWidth',2)
81     hold off
82     axis([-1.5 1.5 -1.5 1.5])
83     daspect([1 1 1])
84     Pts_aprox
85     alfa_aprox
86 end
87
88 if(m==3)
89     r1 = alfa_aprox(1,1)/2;
90     r2 = alfa_aprox(2,1)/2;
91     r3 = alfa_aprox(3,1)/2;
92
93     value(1) = pi*(r1);
```

```
94      value(2) = pi*(r2);
95      value(3) = pi*(r3);
96
97      figure(2)
98      hold on
99      p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K.','MarkerSize', ...
            value(1)+fator,'LineWidth',2);
100     p2 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'K.','MarkerSize', ...
            value(2)+fator,'LineWidth',2);
101     p3 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'K.','MarkerSize', ...
            value(3)+fator,'LineWidth',2);
102     p4 = plot(x(1,:),y(1,:),'K','LineWidth',2)
103     hold off
104     axis([-1.5 1.5 -1.5 1.5])
105     daspect([1 1 1])
106     Pts_aprox
107     alfa_aprox
108 end
109
110 if(m==4)
111     r1 = alfa_aprox(1,1)/2;
112     r2 = alfa_aprox(2,1)/2;
113     r3 = alfa_aprox(3,1)/2;
114
115     value(1) = pi*(r1);
116     value(2) = pi*(r2);
117     value(3) = pi*(r3);
118
119     figure(2)
120     hold on
121     p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K.','MarkerSize', ...
            value(1)+fator,'LineWidth',2);
122     p2 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'K.','MarkerSize', ...
            value(2)+fator,'LineWidth',2);
123     p3 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'K.','MarkerSize', ...
            value(3)+fator,'LineWidth',2);
124     p4 = plot(x(1,:),y(1,:),'K','LineWidth',2)
125     hold off
126     axis([-1.5 1.5 -1.5 1.5])
127     daspect([1 1 1])
128     Pts_aprox
129     alfa_aprox
130 end
131 figure(3)
132 hold on
133 p1 = plot(xygr(:,1),xygr(:,2),'k.','LineWidth',2);
134 axis([-1.5 1.5 -1.5 1.5])
```

```
135 p2 = plot(x(1,:),y(1,:),'K','LineWidth',2);
136 daspect([1 1 1])
137 hold off
138
139 toc
```

```
1  %EXEMPLO 3
2  clear all
3  close all
4  clc
5
6  tic
7  %Intensidade das cargas
8  alpha(1,1) = 8;
9  alpha(2,1) = 8;
10 alpha(3,1) = 8;
11
12 %Localizacoes das cargas
13 % xyst(1,1) =   0.7296
14 % xyst(1,2) =   0.3095
15 %
16 % xyst(2,1) =   -0.3816
17 % xyst(2,2) =   0.2655
18 %
19 % xyst(3,1) =  0.4663
20 % xyst(3,2) =   -0.5478
21
22 %Tamanho do grid
23 tam = 100;
24 [p1 p2] = sunflower_2D(tam,0);
25 p = [p1 p2];
26
27 pto1=63;
28 pto2=22;
29 pto3=52;
30
31 xyst(1,1) =   p(pto1,1);
32 xyst(1,2) =   p(pto1,2);
33
34 xyst(2,1) =   p(pto2,1);
35 xyst(2,2) =   p(pto2,2);
36
37 xyst(3,1) =   p(pto3,1);
38 xyst(3,2) =   p(pto3,2);
39
40 %Variavel auxiliar
```

```
41 fator = 30;
42
43 %ruido (em porcentagem)
44 noise = 0;
45 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainLap_circular(alpha, xyst, noise, tam);
46 alfa_aprox1 = alfa_aprox;
47 Pts_aprox1 = Pts_aprox;
48 M = size(q_star,1);
49 z = linspace(0,1,M);
50 [x,y] = cylinder(1,1000);
51
52 r1 = alfa_aprox(1,1)/2;
53 r2 = alfa_aprox(2,1)/2;
54 r3 = alfa_aprox(3,1)/2;
55
56 value(1) = pi*(r1);
57 value(2) = pi*(r2);
58 value(3) = pi*(r3);
59 figure(2)
60 hold on
61 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
       value(1) + fator,'LineWidth',2);
62 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
       value(2) + fator ,'LineWidth',2);
63 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
       value(3)+ fator ,'LineWidth',2);
64 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
65
66 hold off
67 axis([-1.5 1.5 -1.5 1.5])
68 figure(3)
69 plot(z,q_star,'k.-',z,q_star_noise,'k*');
70 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
71 legend('without noise','with noise')
72
73 noise = 5;
74 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainLap_circular(alpha, xyst, noise, tam);
75 alfa_aprox2 = alfa_aprox;
76 Pts_aprox2 = Pts_aprox;
77 r1 = alfa_aprox(1,1)/2;
78 r2 = alfa_aprox(2,1)/2;
79 r3 = alfa_aprox(3,1)/2;
80
```

```
value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(4)
hold on
p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
    value(1) + fator,'LineWidth',2);
p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
    value(2) + fator ,'LineWidth',2);
p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
    value(3)+ fator ,'LineWidth',2);
p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)

hold off
axis([-1.5 1.5 -1.5 1.5])

figure(5)
plot(z,q_star,'k.-',z,q_star_noise,'k*');
title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
    ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
    ', ', num2str(alfa_aprox(3,1))])
legend('without noise','with noise')

noise = 10;
[alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
    mainLap_circular(alpha, xyst, noise, tam);
alfa_aprox3 = alfa_aprox;
Pts_aprox3 = Pts_aprox;
r1 = alfa_aprox(1,1)/2;
r2 = alfa_aprox(2,1)/2;
r3 = alfa_aprox(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(6)
hold on
p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
    value(1) + fator,'LineWidth',2);
p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
    value(2) + fator ,'LineWidth',2);
p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
    value(3)+ fator ,'LineWidth',2);
p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)

hold off
axis([-1.5 1.5 -1.5 1.5])
```

```
119
120 figure(7)
121 plot(z,q_star,'k.-',z,q_star_noise,'k*');
122 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
123 legend('without noise','with noise')
124
125 noise = 20;
126 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainLap_circular(alpha, xyst, noise, tam);
127 alfa_aprox4 = alfa_aprox;
128 Pts_aprox4 = Pts_aprox;
129 r1 = alfa_aprox(1,1)/2;
130 r2 = alfa_aprox(2,1)/2;
131 r3 = alfa_aprox(3,1)/2;
132
133 value(1) = pi*(r1);
134 value(2) = pi*(r2);
135 value(3) = pi*(r3);
136 figure(8)
137 hold on
138 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
       value(1) + fator,'LineWidth',2);
139 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
       value(2) + fator ,'LineWidth',2);
140 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
       value(3)+ fator ,'LineWidth',2);
141
142 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
143
144 hold off
145 axis([-1.5 1.5 -1.5 1.5])
146
147 figure(9)
148 plot(z,q_star,'k.-',z,q_star_noise,'k*');
149 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
150 legend('without noise','with noise')
151
152 noise = 40;
153 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainLap_circular(alpha, xyst, noise, tam);
154 alfa_aprox5 = alfa_aprox;
155 Pts_aprox5 = Pts_aprox;
156 r1 = alfa_aprox(1,1)/2;
```

```
157 r2 = alfa_aprox(2,1)/2;
158 r3 = alfa_aprox(3,1)/2;
159
160 value(1) = pi*(r1);
161 value(2) = pi*(r2);
162 value(3) = pi*(r3);
163 figure(10)
164 hold on
165 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
       value(1) + fator,'LineWidth',2);
166 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
       value(2) + fator ,'LineWidth',2);
167 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
       value(3)+ fator ,'LineWidth',2);
168 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
169
170 hold off
171 axis([-1.5 1.5 -1.5 1.5])
172
173 figure(11)
174 plot(z,q_star,'k.-',z,q_star_noise,'k*');
175 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
176 legend('without noise','with noise')
177
178 toc
```

```
1  %AJUSTES
2  clear all
3  close all
4  clc
5
6  tic
7
8  %Tamanho do grid
9  L_Grid = 100;
10
11 %Intensidade das cargas
12 alpha(1,1) = 10;
13
14 %Grid gerado com sunflower seed
15 [p1 p2] = sunflower_2D(L_Grid,0);
16 p = [p1 p2];
17 %lambda1
18 % pto=7;
```

```
19
20 %lambda2
21 % pto=35;
22
23 %lambda3
24 pto=50;
25
26 dx_dy=0;
27 %Localizacoes das cargas
28
29 xyst(1,1) =   p(pto,1)+dx_dy;
30 xyst(1,2) =   p(pto,2)-dx_dy;
31
32 R = 1.1 : 0.1 : 3;
33 L =  length(R)
34 aux=1;
35 for i=1:L
36     [alfa_aprox, Pts_aprox, xyzc, xyzs, xyzgr] = ...
          mainLap_2D(alpha, xyst, L_Grid, R(i));
37     erroABS_alpha(aux) = abs(alpha - alfa_aprox);
38     erroREL_alpha(aux) = erroABS_alpha(aux) / abs(alpha);
39     distEUC(aux) = norm(Pts_aprox - xyst);
40     aux = aux+1;
41 end
42
43 figure(1)
44 plot(R, 100*erroREL_alpha,'b.-','MarkerSize', ...
      10,'LineWidth',1,'LineStyle','--')
45 xlabel('Raio R') % x-axis label
46 ylabel('Erro Relativo de \alpha (%)','FontSize',20) % y-axis label
47 title('Curva de Erro','FontSize',20)
48
49 figure(2)
50 plot(R, distEUC,'b.-','MarkerSize', ...
      10,'LineWidth',1,'LineStyle','--')
51 xlabel('Raio R','FontSize',20) % x-axis label
52 ylabel('Norma Euclidiana ||x - x^*||','FontSize',20) % y-axis label
53 tic
```

Equação do Tipo Helmholtz Bidimensional

Programa Principal

```
1 %Ex. 4
```

```
% function [valor_alfa, Pts_otimo, xyc, xys, xygr] = ...
    mainHelm_2D(alpha, xyst, L_Grid,lambda,R)
%Ex. 5
% function [valor_alfa, Pts_otimo, xyc, xys, xygr] = ...
    mainHelm_2D(alpha, xyst, L_Grid)
%Ex. 6
% function [valor_alfa, Pts_otimo, u_star, q_star, xygr,M,N] = ...
    mainHelm_2D(alpha, xyst,L_Grid,m)
% Ex. 7 (Ex. 12)
function [valor_alfa, Pts_otimo, q_star_without_noise, ...
    q_star_with_noise, xyc, xys, xygr] = mainHelm_2D(alpha, ...
    xyst, noise, L_Grid)

lambda = + 9.5;
% lambda = + 1;
% lambda = - 4;
%Tolerancia para a funcao lsqr
tol=1e-15;
%Raio do dominio
r = 1.0;
%Raio dos pontos fonte
R = 3.0;
% R = 2.4
%Numero de pontos de colocacao
M = 15;

% M = 50;
%Numero de pontos de fonte
N = 12;

% N = 20;
% Pontos fonte e de colocacao
xyc = zeros(M,2);
xys = zeros(N,2);

%vetor normal
nxye = xyc;

for k = 1:M
   nxye(k,1) = cos(2*pi*k/M);
   nxye(k,2) = sin(2*pi*k/M);
end
xyc(:,1) = r*nxye(:,1);
xyc(:,2) = r*nxye(:,2);

for k = 1:N
   xys(k,1) = R*cos(2*pi*k/N);
```

```
44      xys(k,2) = R*sin(2*pi*k/N);
45  end
46
47  %numero de cargas para serem reconstruidas
48  numPtsfontcurr = size(alpha,1);
49  %Numero de pontos para integracao de linha
50  MT = 25;
51  %pontos para integracao de linha
52  xy_int = zeros(MT,2);
53
54  for k = 1:MT
55      xy_int(k,1) = r*cos(2*pi*k/MT);
56      xy_int(k,2) = r*sin(2*pi*k/MT);
57  end
58
59  % Dados de Cauchy
60  [q_star, u_star] = ...
        calc_q_star_hel_2D(xys,M,alpha,xyst,r,numPtsfontcurr,lambda);
61  %Gera (u*,q*) associado a uma fonte distribuida de raio rd
62  %[q_star, u_star] = calc_q_star_hel_2D_dist(r,lambda);
63  % acrescenta ruido
64  q_star_without_noise = q_star;
65  q_star_with_noise = q_star.*(1+ noise*(2*rand(M,1)-1)/100 );
66  q_star = q_star_with_noise;
67
68  %%
69  %Calcular os coeficientes de uD
70  M_uD=[];
71  for i=1:M
72      for j=1:N
73          if(lambda < 0)
74              %Equacao de Helmholtz
75              dist2(i,j) = (xyc(i,1)-xys(j,1))^2 + ...
                    (xyc(i,2)-xys(j,2))^2;
76              arg = sqrt(-lambda)*sqrt(dist2(i,j));
77              H0 = besselh(0,arg);
78              M_uD(i,j) = real(1i*H0)/4;
79          else
80              %Equacao de Helmholtz modificada
81              dist2(i,j) = (xyc(i,1)-xys(j,1))^2 + ...
                    (xyc(i,2)-xys(j,2))^2;
82              arg = sqrt(lambda)*sqrt(dist2(i,j));
83              K0 = besselk(0,arg);
84              M_uD(i,j) = K0/(2*pi);
85          end
86      end
87      vetor_uD(i,1) = u_star(i);
```

```
88  end
89  %coeficientes de uD
90  c_uD = lsqr(M_uD,vetor_uD,tol);
91  %calcula os coeficientes de uN
92  M_uN=[];
93  for i=1:M
94      for j=1:N
95          if(lambda < 0)
96              %Equacao de Helmholtz
97              dist2(i,j) = (xyc(i,1)-xys(j,1))^2 + ...
                    (xyc(i,2)-xys(j,2))^2;
98              arg = sqrt(-lambda)*sqrt(dist2(i,j));
99              H1 = besselh(1,arg);
100             %Derivada normal de uN
101             DuN_x(i,j) = -real( ( (xyc(i,1) - ...
                    xys(j,1))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(dist2(i,j)) );
102             DuN_y(i,j) = -real( ( (xyc(i,2) - ...
                    xys(j,2))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(dist2(i,j)) );
103             M_uN(i,j)  = DuN_x(i,j)*nxye(i,1) + ...
                    DuN_y(i,j)*nxye(i,2);
104         else
105             %Equacao de Helmholtz modificada
106             dist2(i,j) = (xyc(i,1)-xys(j,1))^2 + ...
                    (xyc(i,2)-xys(j,2))^2;
107             arg = sqrt(lambda)*sqrt(dist2(i,j));
108             K1 = besselk(1,arg);
109             DuN_x(i,j) = -( (xyc(i,1)-xys(j,1))*K1*sqrt(lambda) ...
                    )/( 2*pi*sqrt(dist2(i,j)) );
110             DuN_y(i,j) = -( (xyc(i,2)-xys(j,2))*K1*sqrt(lambda) ...
                    )/( 2*pi*sqrt(dist2(i,j)) );
111             M_uN(i,j)  = DuN_x(i,j)*nxye(i,1) + ...
                    DuN_y(i,j)*nxye(i,2);
112         end
113     end
114     vetor_uN(i,1) = - q_star(i);
115 end
116 %coeficientes de uN
117 c_uN = lsqr(M_uN,vetor_uN,tol);
118 %%
119 %calcula uD e uN nos na fronteira auxiliando na integracao numerica
120 matriz_uD_uN = [];
121 dist2=[];
122 for i=1:MT
123     for j=1:N
124         if(lambda < 0)
```

```
125             dist2(i,j) = (xy_int(i,1)-xys(j,1))^2 + ...
                    (xy_int(i,2)-xys(j,2))^2;
126             arg = sqrt(-lambda)*sqrt(dist2(i,j));
127             H0 = besselh(0,arg);
128             matriz_uD_uN(i,j) = real(1i*H0)/4;
129         else
130             dist2(i,j) = (xy_int(i,1)-xys(j,1))^2 + ...
                    (xy_int(i,2)-xys(j,2))^2;
131             arg = sqrt(lambda)*sqrt(dist2(i,j));
132             K0  = besselk(0,arg);
133             matriz_uD_uN(i,j) = K0/(2*pi);
134         end
135     end
136 end
137 uD_int = matriz_uD_uN*c_uD;
138 uN_int = matriz_uD_uN*c_uN;
139 %%
140 tam = L_Grid;
141 [xgr ygr] = sunflower_2D(tam,0);
142 %Pontos do grid
143 xygr = [ xgr ygr ];
144
145 %hi: valor aproximado de hi nos pontos da integracao de linha
146 %di: valor aproximado de hi(uD-uN) nos pontos da integracao de linha
147 [hi di] = calc_hi_di_hel_2D(...
148                     xyc,nxye,xys,xy_int,...
149                     xygr,uD_int,uN_int,...
150                     r,tol,...
151                     lambda,tam);
152
153 aux = 1;
154 %Numero de cargas a serem reconstruidas
155 % numPtsfontcurr = m;
156 switch numPtsfontcurr
157     %reconstrucao de uma cargas
158     case 1
159         for i1 = 1:tam
160             entry_hi = hi(:,i1);
161             entry_di = di(:,i1);
162             %Calcula a matriz H e o vetor d
163             [H d] = calc_H_d_hel_2D(entry_hi, ...
                    entry_di,numPtsfontcurr,MT,r);
164             alfas(:,aux) = lsqr(H,d,tol);
165             %Armazena os valores da variacao do funcional
166             funcional(aux,1)=-0.5*alfas(:,aux)'*d;
167             aux_index(:,aux) = i1;
168             aux = aux + 1;
```

```
169         end
170     %reconstrucao de duas cargas
171     case 2
172         for i1 = 1:tam
173             for i2 = i1:tam
174                 entry_hi = [ hi(:,i2) hi(:,i1) ];
175                 entry_di = [ di(:,i2) di(:,i1) ];
176                 [H d] = calc_H_d_hel_2D(entry_hi, ...
                        entry_di,numPtsfontcurr,MT,r);
177 
178                 alfas(:,aux) = lsqr(H,d,tol);
179                 funcional(aux,1)=-0.5*alfas(:,aux)'*d;
180                 aux_index(:,aux) = [i2 i1];
181                 aux = aux + 1;
182             end
183         end
184     %reconstrucao de tres cargas
185     case 3
186         for i1 = 1:tam
187             for i2 = i1:tam
188                 for i3 = i2:tam
189                     entry_hi = [ hi(:,i3) hi(:,i2) hi(:,i1)];
190                     entry_di = [ di(:,i3) di(:,i2) di(:,i1)];
191                     [H d] = calc_H_d_hel_2D(entry_hi, ...
                            entry_di,numPtsfontcurr,MT,r);
192 
193                     alfas(:,aux) = lsqr(H,d,tol);
194                     funcional(aux,1)=-0.5*alfas(:,aux)'*d;
195                     aux_index(:,aux) = [i3 i2 i1];
196                     aux = aux + 1;
197                 end
198             end
199         end
200     %reconstrucao de quatro cargas
201     case 4
202         for i1 = 1:tam
203             for i2 = i1:tam
204                 for i3 = i2:tam
205                     for i4 = i3:tam
206                         entry_hi = [ hi(:,i3) hi(:,i2) hi(:,i1) ...
                                hi(:,i4)];
207                         entry_di = [ di(:,i3) di(:,i2) di(:,i1) ...
                                di(:,i4)];
208                         [H d] = calc_H_d_hel_2D(entry_hi, ...
                                entry_di,numPtsfontcurr,MT,r);
209 
210                         alfas(:,aux) = lsqr(H,d,tol);
```

```
211                         funcional(aux,1)=-0.5*alfas(:,aux)'*d;
212                         aux_index(:,aux) = [i3 i2 i1 i4];
213                         aux = aux + 1;
214                     end
215                 end
216             end
217         end
218     otherwise
219 end
220
221 %retorna o valor minimo da variacao do funcional
222
223 [value, index] = min(funcional);
224 funcional  = value;
225 valor_alfa = alfas(:,index);
226 Pts_otimo = xygr(aux_index(:,index),:);
```

Rotinas Utilizadas

```
1  %Gera os par de dados de Cauchy (u*, q* )associados a b*
2  function [q_star, u_star]=calc_q_star_hel_2D(xys,M,alpha,xyst,...
3      r,numPtsfontcurr,lambda)
4  %Pontos de colocacao
5
6  xy = zeros(M,2);
7  for k = 1:M
8     xy(k,1) = cos(2*pi*k/M);
9     xy(k,2) = sin(2*pi*k/M);
10 end
11 xyc(:,1) = r*xy(:,1);
12 xyc(:,2) = r*xy(:,2);
13 %Vetor normal
14 nxye = xy;
15 %Numero de pontos fonte
16 [l c] = size(xys);
17 N = l;
18 %Dado de Dirichlet prescrito
19 u_star = xy(:,1);
20 %Calcula a derivada normal e a matriz do sistema
21 for i = 1:M
22     %Parcela da solucao aproximada do somatorio
23     for j = 1:N
24         if(lambda < 0)
25             %Equacao de Helmholtz
26             M_qst_H(i,j) = (xyc(i,1) - xys(j,1))^2 + (xyc(i,2) - ...
                   xys(j,2))^2;
```

```
27              arg = sqrt(-lambda)*sqrt(M_qst_H(i,j));
28              H1 = besselh(1,arg);
29              Gx_H(i,j) = -real( ( (xyc(i,1) - ...
                    xys(j,1))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(M_qst_H(i,j)) );
30              Gy_H(i,j) = -real( ( (xyc(i,2) - ...
                    xys(j,2))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(M_qst_H(i,j)) );
31              AN(i,j) = Gx_H(i,j)*nxye(i,1) + Gy_H(i,j)*nxye(i,2);
32          else
33              %Equacao de Helmholtz modificada
34              M_qst_H(i,j) = (xyc(i,1) - xys(j,1))^2 + (xyc(i,2) - ...
                    xys(j,2))^2;
35              arg = sqrt(lambda)*sqrt(M_qst_H(i,j));
36              K1 = besselk(1,arg);
37              Gx_H(i,j) =  -( (xyc(i,1) - ...
                    xys(j,1))*K1*sqrt(lambda) )/( ...
                    2*pi*sqrt(M_qst_H(i,j)) );
38              Gy_H(i,j) =  -( (xyc(i,2) - ...
                    xys(j,2))*K1*sqrt(lambda) )/( ...
                    2*pi*sqrt(M_qst_H(i,j)) );
39              AN(i,j) = Gx_H(i,j)*nxye(i,1) + Gy_H(i,j)*nxye(i,2);
40          end
41      end
42      %vetor do sistema
43      fun(i,1)= u_star(i);
44      q_star2(i,1) = 0;
45      for k=1:numPtsfontcurr
46          if(lambda < 0)
47              M_qst(i,1,k) = (xyc(i,1) - xyst(k,1))^2 + (xyc(i,2) ...
                    - xyst(k,2))^2;
48              arg = sqrt(-lambda)*sqrt(M_qst(i,1,k));
49              H0 = besselh(0,arg);
50              fun(i,1) = fun(i,1) - (alpha(k,1)*real(1i*H0))/4;
51              %Parcela da solucao aproximada associada ao alpha_i
52              H1 = besselh(1,arg);
53              Gx(i,1,k) = - real( ( (xyc(i,1) - ...
                    xyst(k,1))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(M_qst(i,1,k)) );
54              Gy(i,1,k) = - real( ( (xyc(i,2) - ...
                    xyst(k,2))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(M_qst(i,1,k)) );
55              q_star2(i,1) = q_star2(i,1) + alpha(k,1)*( ...
                    Gx(i,1,k)*nxye(i,1) + Gy(i,1,k)*nxye(i,2) );
56          else
57              M_qst(i,1,k) = (xyc(i,1) - xyst(k,1))^2 + (xyc(i,2) ...
                    - xyst(k,2))^2;
```

```
58              arg = sqrt(lambda)*sqrt(M_qst(i,1,k));
59              K0 = besselk(0,arg);
60              fun(i,1) = fun(i,1) - alpha(k,1)*K0/(2*pi);
61              K1 = besselk(1,arg);
62              Gx(i,1,k) =  -( (xyc(i,1) - ...
                    xyst(k,1))*K1*sqrt(lambda) )/( ...
                    2*pi*sqrt(M_qst(i,1,k)) );
63              Gy(i,1,k) =  -( (xyc(i,2) - ...
                    xyst(k,2))*K1*sqrt(lambda) )/( ...
                    2*pi*sqrt(M_qst(i,1,k)) );
64              q_star2(i,1) = q_star2(i,1) + alpha(k,1)*( ...
                    Gx(i,1,k)*nxye(i,1) + Gy(i,1,k)*nxye(i,2) );
65          end
66      end
67  end
68  if(lambda < 0)
69      arg = sqrt(-lambda)*sqrt(M_qst_H);
70      H0 = besselh(0,arg);
71      AD = real(1i*H0)/4;
72      cD = lsqr(AD,fun);
73      q_star1=real(AN)*cD;
74      %Correspondente dado de Newmann q*
75      q_star = -( q_star1 + q_star2 );
76  else
77      arg = sqrt(lambda)*sqrt(M_qst_H);
78      K0 = besselk(0,arg);
79      AD = K0/(2*pi);
80      cD = lsqr(AD,fun);
81      q_star1=AN*cD;
82      q_star = -( q_star1 + q_star2 );
83  end
```

```
1  %As variaveis hi e di auxiliam no calculo da matriz H e o vetor d
2  function [hi di] = calc_hi_di_hel_2D(...
3                              xyc,nxye,xys,xy_int,...
4                              pts_gr,u_star_integral,uN,...
5                              r,tol,lambda,numPts)
6  %%
7  %Numero de pontos de colocacao
8  [l1 c1] = size(xyc);
9  M = l1;
10 %Numero de pontos fonte
11 [l2 c2] = size(xys);
12 N = l2;
13 %Numero de pontos para a integracao de linha
14 [l3 c3] = size(xy_int);
```

```
MT = l3;

matriz_vi = [];
for i=1:M
    for j=1:N
         if(lambda < 0)
            %Equacao de Helmholtz
            raio2 = (xyc(i,1) - xys(j,1))^2 + (xyc(i,2) - ...
                xys(j,2))^2;
            arg = sqrt(-lambda)*sqrt(raio2);
            H0 = besselh(0,arg);
            %Matriz utilizada para calcular os coeficientes de vi
            matriz_vi(i,j) = real(1i*H0)/4;
            H1 = besselh(1,arg);
            %Matriz utilizada para calcular a derivada normal de vi
            %Tambem sera utilizada como matriz de hi
            matriz_Dvi1_hiH_x(i,j) = -real( ( (xyc(i,1) - ...
                xys(j,1))*sqrt(-lambda)*(1i*H1)) )/( ...
                4*sqrt(raio2) );
            matriz_Dvi1_hiH_y(i,j) = -real( ( (xyc(i,2) - ...
                xys(j,2))*sqrt(-lambda)*(1i*H1)) )/( ...
                4*sqrt(raio2) );
            matriz_Dvi1_hiH(i,j)   = ...
                matriz_Dvi1_hiH_x(i,j)*nxye(i,1) + ...
                matriz_Dvi1_hiH_y(i,j)*nxye(i,2);
         else
            %Equacao de Helmholtz modificada
            raio2 = (xyc(i,1) - xys(j,1))^2 + (xyc(i,2) - ...
                xys(j,2))^2;
            arg = sqrt(lambda)*sqrt(raio2);
            K0  = besselk(0,arg);
            matriz_vi(i,j) = K0/(2*pi);
            K1  = besselk(1,arg);
            matriz_Dvi1_hiH_x(i,j) = -( (xyc(i,1) - ...
                xys(j,1))*K1*sqrt(lambda) )/( 2*pi*sqrt(raio2) );
            matriz_Dvi1_hiH_y(i,j) = -( (xyc(i,2) - ...
                xys(j,2))*K1*sqrt(lambda) )/( 2*pi*sqrt(raio2) );
            matriz_Dvi1_hiH(i,j)   = ...
                matriz_Dvi1_hiH_x(i,j)*nxye(i,1) + ...
                matriz_Dvi1_hiH_y(i,j)*nxye(i,2);
         end
        end
    %Calcula a derivada normal da solucao particular de vi
    for k = 1:numPts
        if(lambda < 0)
            %Equacao de Helmholtz
            raio2 = (xyc(i,1) - pts_gr(k,1))^2 + (xyc(i,2) - ...
```

```
                pts_gr(k,2))^2;
50              arg = sqrt(-lambda)*sqrt(raio2);
51              H0 = besselh(0,arg);
52              %Vetor vi utiliado para calcular os coeficientes de vi
53              vetor_vi(i,1,k) = -real(1i*H0)/4;
54              H1 = besselh(1,arg);
55              Dvi2_x(i,1,k) =  -real( ( (xyc(i,1) - ...
                    pts_gr(k,1))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(raio2) );
56              Dvi2_y(i,1,k) =  -real( ( (xyc(i,2) - ...
                    pts_gr(k,2))*sqrt(-lambda)*(1i*H1)) )/( ...
                    4*sqrt(raio2) );
57              Dvi2(i,1,k)   = Dvi2_x(i,1,k)*nxye(i,1) + ...
                    Dvi2_y(i,1,k)*nxye(i,2);
58          else
59              %Equacao de Helmholtz modificada
60              raio2 = (xyc(i,1) - pts_gr(k,1))^2 + (xyc(i,2) - ...
                    pts_gr(k,2))^2;
61              arg = sqrt(lambda)*sqrt(raio2);
62              K0 = besselk(0,arg);
63              vetor_vi(i,1,k) = -K0/(2*pi);
64              K1 = besselk(1,arg);
65              Dvi2_x(i,1,k) =  -( (xyc(i,1) - ...
                    pts_gr(k,1))*K1*sqrt(lambda) )/( ...
                    2*pi*sqrt(raio2) );
66              Dvi2_y(i,1,k) =  -( (xyc(i,2) - ...
                    pts_gr(k,2))*K1*sqrt(lambda) )/( ...
                    2*pi*sqrt(raio2) );
67              Dvi2(i,1,k)   = Dvi2_x(i,1,k)*nxye(i,1) + ...
                    Dvi2_y(i,1,k)*nxye(i,2);
68          end
69      end
70  end
71  %Coeficientes de vi
72  for k=1:numPts
73      cvi(:,k) = lsqr(matriz_vi,vetor_vi(:,1,k),tol);
74      Dvi1(:,k) = matriz_Dvi1_hiH*cvi(:,k);
75      Dvi(:,k) = Dvi1(:,k) + Dvi2(:,1,k);
76      vetor_hiH(:,1,k) = - Dvi(:,k);
77  end
78  %coeficientes de hi
79  for k=1:numPts
80      chi(:,k) = lsqr(matriz_Dvi1_hiH,vetor_hiH(:,1,k),tol);
81  end
82  %%
83  %calcular as variaveis auxiliares hi e di
84  for i=1:MT
```

```
85      for j=1:N
86          if(lambda < 0)
87              %Equacao de Helmholtz
88              raio2 = (xy_int(i,1) - xys(j,1))^2 + (xy_int(i,2) - ...
                    xys(j,2))^2;
89              arg = sqrt(-lambda)*sqrt(raio2);
90              H0 = besselh(0,arg);
91              matriz_hiH(i,j) = real(1i*H0)/4;
92          else
93              %Equacao de Helmholtz modificada
94              raio2 = (xy_int(i,1) - xys(j,1))^2 + (xy_int(i,2) - ...
                    xys(j,2))^2;
95              arg = sqrt(lambda)*sqrt(raio2);
96              K0  = besselk(0,arg);
97              matriz_hiH(i,j) = K0/(2*pi);
98          end
99      end
100     for k=1:numPts
101         hi(i,k) = matriz_hiH(i,:)*chi(:,k);
102         di(i,k) = hi(i,k)*(u_star_integral(i) - uN(i));
103     end
104 end
```

```
1  %Calcula a matriz H e o vetor d com base nas variaveis hi e di
2  function [H d] = calc_H_d_hel_2D(hi, di,numPtsfontcurr,MT,r)
3  for i = 1:numPtsfontcurr
4      for j=i:numPtsfontcurr
5          if (i==j)
6              funHij = hi(:,i).^2;
7              H(i,j) = T_Simpson_2D(funHij, 2*pi*r/MT);
8          else
9              funHij = hi(:,i).*hi(:,j);
10             H(i,j) = T_Simpson_2D(funHij, 2*pi*r/MT);
11             H(j,i) = H(i,j);
12         end
13     end
14     d(i,1) = T_Simpson_2D(di(:,i), 2*pi*r/MT);
15 end
```

```
1  function [x y] = sunflower_2D(n, alpha)
2      % number of boundary points
3      b = round(alpha*sqrt(n));
4      % golden ratio
5      phi = (sqrt(5)+1)/2;
```

```
    for k=1:n
        r = radius(k,n,b);
        theta = 2*pi*k/phi^2;
        x(k,1) = r*cos(theta);
        y(k,1) = r*sin(theta);
    end
end
function r = radius(k,n,b)
    if k>n-b
        % put on the boundary
        r = 1;
    else
        % apply square root
        r = sqrt(k-1/2)/sqrt(n-(b+1)/2);
    end
end
```

```
function res=T_Simpson_2D(u, h)
n=length(u)-1;
aux1=0;
aux2=0;
for i=1:n/2
    index=2*i-1;
    aux1=aux1+4*u(index+1,1);
end
for i=1:n/2-1
    index=2*i;
    aux2=aux2+2*u(index+1,1);
end
res=(h*(u(1,1)+u(n,1)+aux1+aux2) )/3;
```

```
function J = ...
    integralKV(m,alfa_aprox,Pts_aprox,u_star,q_star,M,N,R,r,lambda)
xy = zeros(M,2);
for k = 1:M
   xy(k,1) = cos(2*pi*k/M);
   xy(k,2) = sin(2*pi*k/M);
end
%Vetor normal
nxye = xy;

xyc(:,1) = r*xy(:,1);
xyc(:,2) = r*xy(:,2);

```

```
13 for k = 1:N
14    xy(k,1) = cos(2*pi*k/N);
15    xy(k,2) = sin(2*pi*k/N);
16 end
17 xys(:,1) = R*xy(:,1);
18 xys(:,2) = R*xy(:,2);
19 %Calcula a derivada normal e a matriz do sistema
20 for i = 1:M
21     %Parcela da solucao aproximada do somatorio
22     for j = 1:N
23         %Equacao de Helmholtz modificada
24         M_qst_H(i,j) = (xyc(i,1) - xys(j,1))^2 + (xyc(i,2) - ...
               xys(j,2))^2;
25         arg = sqrt(lambda)*sqrt(M_qst_H(i,j));
26         K1 = besselk(1,arg);
27         Gx_H(i,j) =  -( (xyc(i,1) - xys(j,1))*K1*sqrt(lambda) ...
               )/( 2*pi*sqrt(M_qst_H(i,j)) );
28         Gy_H(i,j) =  -( (xyc(i,2) - xys(j,2))*K1*sqrt(lambda) ...
               )/( 2*pi*sqrt(M_qst_H(i,j)) );
29         AN(i,j) = Gx_H(i,j)*nxye(i,1) + Gy_H(i,j)*nxye(i,2);
30     end
31     %vetor do sistema
32     fun(i,1)= - q_star(i);
33     for k=1:m
34         M_qst(i,1,k) = (xyc(i,1) - Pts_aprox(k,1))^2 + (xyc(i,2) ...
               - Pts_aprox(k,2))^2;
35         arg = sqrt(lambda)*sqrt(M_qst(i,1,k));
36         K1 = besselk(1,arg);
37         Gx(i,1,k) =  -( (xyc(i,1) - ...
               Pts_aprox(k,1))*K1*sqrt(lambda) )/( ...
               2*pi*sqrt(M_qst(i,1,k)) );
38         Gy(i,1,k) =  -( (xyc(i,2) - ...
               Pts_aprox(k,2))*K1*sqrt(lambda) )/( ...
               2*pi*sqrt(M_qst(i,1,k)) );
39         fun(i,1) = fun(i,1) - alfa_aprox(k,1)*( ...
               Gx(i,1,k)*nxye(i,1) + Gy(i,1,k)*nxye(i,2) );
40     end
41 end
42 switch m
43     case 1
44         cN = lsqr(AN,fun);
45         arg1 = sqrt(lambda)*sqrt(M_qst_H);
46         K0 = besselk(0,arg1);
47         A1 = K0/(2*pi);
48         arg2 = sqrt(lambda)*sqrt(M_qst);
49         K0 = besselk(0,arg2);
50         A2 = K0/(2*pi);
```

```
51          uN=A1*cN + A2*alfa_aprox;
52      case 2
53          cN = lsqr(AN,fun);
54          arg1 = sqrt(lambda)*sqrt(M_qst_H);
55          K0 = besselk(0,arg1);
56          A1 = K0/(2*pi);
57          arg2 = sqrt(lambda)*sqrt(M_qst);
58          K0 = besselk(0,arg2);
59          A2 = K0/(2*pi);
60          A3 = [A2(:,1,1) A2(:,1,2)];
61          uN=A1*cN + A3*alfa_aprox;
62      case 3
63          cN = lsqr(AN,fun);
64          arg1 = sqrt(lambda)*sqrt(M_qst_H);
65          K0 = besselk(0,arg1);
66          A1 = K0/(2*pi);
67          arg2 = sqrt(lambda)*sqrt(M_qst);
68          K0 = besselk(0,arg2);
69          A2 = K0/(2*pi);
70          A3 = [A2(:,:,1) A2(:,:,2) A2(:,:,3)];
71          uN=A1*cN + A3*alfa_aprox;
72      case 4
73          cN = lsqr(AN,fun);
74          arg1 = sqrt(lambda)*sqrt(M_qst_H);
75          K0 = besselk(0,arg1);
76          A1 = K0/(2*pi);
77          arg2 = sqrt(lambda)*sqrt(M_qst);
78          K0 = besselk(0,arg2);
79          A2 = K0/(2*pi);
80          A3 = [A2(:,:,1) A2(:,:,2) A2(:,:,3) A2(:,:,4)];
81          uN=A1*cN + A3*alfa_aprox;
82      otherwise
83  end
84  %Calcular a integral sobre a fronteira de (uD - uN), em que b_0 ...
        = b*
85  h = 2*pi*r/M;
86  J = 0.5*T_Simpson_2D((u_star - uN).^2, h);
```

Exemplos Numéricos

```
1  %EXEMPLO 4
2  clear all
3  close all
4  clc
5
6  tic
```

```
%Tamanho do grid
L_Grid = 100;
%Intensidade da fonte
alpha(1,1) = 10;
% alpha(1,1) = 20;
%Grid gerado com sunflower seed
[p1 p2] = sunflower_2D(L_Grid,0);
p = [p1 p2];
% lambda1
pto=7;
% lambda2
%pto=35;
% pto=1;
% lambda3
% pto=50;
%Δ x e Δ y
dx_dy=0.05;
%Localizacao exata de cada carga
xyst(1,1) =   p(pto,1)+dx_dy;
xyst(1,2) =   p(pto,2)-dx_dy;
lambda= +9.5;
% lambda= +1;
% lambda=-4;
%Variacao de R no intervalo I
R = 1.1 : 0.1 : 5;
L =  length(R)
aux=1;
for i=1:L
    [alfa_aprox, Pts_aprox, xyzc, xyzs, xyzgr] = ...
       mainHelm_2D(alpha, xyst, L_Grid, lambda, R(i));
    erroABS_alpha(aux) = abs(alpha - alfa_aprox);
    erroREL_alpha(aux) = erroABS_alpha(aux) / abs(alpha);
    distEUC(aux) = norm(Pts_aprox - xyst);
    aux = aux+1;
end
%Plot dos graficos
figure(1)
plot(R, 100*erroREL_alpha,'b.-','MarkerSize', ...
    10,'LineWidth',1,'LineStyle','--')
xlabel('Raio R')
ylabel('Erro Relativo de \alpha (%)','FontSize',20) % y-axis label
title('Curva de Erro','FontSize',20)

figure(2)
plot(R, distEUC,'b.-','MarkerSize', ...
    10,'LineWidth',1,'LineStyle','--')
xlabel('Raio R','FontSize',20)
```

```
ylabel('Norma Euclidiana ||x - x^*||','FontSize',20)
toc
```

```
%EXEMPLO 5
clear all
close all
clc
tic
%Tamanho do grid
L_Grid = 500;
%Intensidade da fonte
% alpha(1,1) = 10;
alpha(1,1) = 20;
%Grid gerado com sunflower seed
[p1 p2] = sunflower_2D(L_Grid,0);
% p = [p1 p2];
% pto=80;
%Localizacoes das solucoes do problema
% xyst(1,1) =   0.25;
% xyst(1,2) =   -0.32;

xyst(1,1) =   0.14;
xyst(1,2) =   0.48;

% xyst(1,1) =   p(pto,1);
% xyst(1,2) =   p(pto,2);

[alfa_aprox, Pts_aprox, xyc, xys, xygr] = mainHelm_2D(alpha, ...
    xyst, L_Grid);
%Plot dos graficos
figure(1)
hold on
p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'ro ','MarkerSize', ...
    10,'LineWidth',2);
p2 = plot(xyst(1,1),xyst(1,2),'g+','MarkerSize', 10,'LineWidth',2);
p3 = plot(xyc(:,1),xyc(:,2),'K.','MarkerSize', 10,'LineWidth',2);
hold off
legend([p1,p2, p3],'P_{1}Aprox','P_{1}Exa', 'Colloc. points');
axis([-1.5 1.5 -1.5 1.5])
title(['Computed alpha: ',num2str(alfa_aprox(1,1))])
daspect([1 1 1])

alpha
alfa_aprox
Pts_aprox
erro_loc = norm(xyst - Pts_aprox)
```

```
erro_int = abs(alpha - alfa_aprox)/abs(alpha)
toc
```

```
%EXEMPLO 6
clear all
close all
clc

tic
%Tamanho do grid
L_Grid = 100;
%intensidade das cargas
% alpha(1,1) = 6;
% alpha(2,1) = 6;
% alpha(3,1) = 6;

alpha(1,1) = 5;
alpha(2,1) = 10;
alpha(3,1) = 15;

%Grid gerado com sunflower seed
[p1 p2] = sunflower_2D(L_Grid,0);
p = [p1 p2]

% pto1=35;
% pto2=33;
% pto3=50;

pto1=26;
pto2=25;
pto3=53;

xyst(1,1) =   p(pto1,1);
xyst(1,2) =   p(pto1,2);

xyst(2,1) =   p(pto2,1);
xyst(2,2) =   p(pto2,2);

xyst(3,1) =   p(pto3,1);
xyst(3,2) =   p(pto3,2);

%Determina quantas cargas serao reconstruidas
m = 1;
%Variavel auxiliar
fator = 30;

```

```
[x,y,z] = cylinder(1,1000);
[alfa_aprox, Pts_aprox, u_star, q_star, xygr, M, N] = ...
    mainHelm_2D(alpha, xyst, L_Grid,m);
r1 = alpha(1,1)/2;
r2 = alpha(2,1)/2;
r3 = alpha(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);

figure(1)
hold on
p1 = plot(xyst(1,1),xyst(1,2),'K.','MarkerSize', ...
    value(1)+fator,'LineWidth',2);
p2 = plot(xyst(2,1),xyst(2,2),'K.','MarkerSize', ...
    value(2)+fator,'LineWidth',2);
p3 = plot(xyst(3,1),xyst(3,2),'K.','MarkerSize', ...
    value(3)+fator,'LineWidth',2);
p4 = plot(x(1,:),y(1,:),'K','LineWidth',2)
daspect([1 1 1])
hold off
axis([-1.5 1.5 -1.5 1.5])

if(m==1)
    r1 = alfa_aprox(1,1)/2;
    value(1) = pi*(r1);

    figure(2)
    hold on
    p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K. ','MarkerSize', ...
        value(1)+fator,'LineWidth',2);
    p2 = plot(x(1,:),y(1,:),'K','LineWidth',2)
    hold off
    axis([-1.5 1.5 -1.5 1.5])
    daspect([1 1 1])
    Pts_aprox
    alfa_aprox
end
if(m==2)
    r1 = alfa_aprox(1,1)/2;
    r2 = alfa_aprox(2,1)/2;

    value(1) = pi*(r1);
    value(2) = pi*(r2);

    figure(2)
```

```
86      hold on
87      p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K.','MarkerSize', ...
            value(1)+fator,'LineWidth',2);
88      p2 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'K.','MarkerSize', ...
            value(2)+fator,'LineWidth',2);
89      p3 = plot(x(1,:),y(1,:),'K','LineWidth',2)
90      hold off
91      axis([-1.5 1.5 -1.5 1.5])
92      daspect([1 1 1])
93      Pts_aprox
94      alfa_aprox
95  end
96  if(m==3)
97      r1 = alfa_aprox(1,1)/2;
98      r2 = alfa_aprox(2,1)/2;
99      r3 = alfa_aprox(3,1)/2;
100
101     value(1) = pi*(r1);
102     value(2) = pi*(r2);
103     value(3) = pi*(r3);
104
105     figure(2)
106     hold on
107     p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K.','MarkerSize', ...
            value(1)+fator,'LineWidth',2);
108     p2 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'K.','MarkerSize', ...
            value(2)+fator,'LineWidth',2);
109     p3 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'K.','MarkerSize', ...
            value(3)+fator,'LineWidth',2);
110     p4 = plot(x(1,:),y(1,:),'K','LineWidth',2)
111     daspect([1 1 1])
112     hold off
113     axis([-1.5 1.5 -1.5 1.5])
114     Pts_aprox
115     alfa_aprox
116 end
117 if(m==4)
118     r1 = alfa_aprox(1,1)/2;
119     r2 = alfa_aprox(2,1)/2;
120     r3 = alfa_aprox(3,1)/2;
121
122     value(1) = pi*(r1);
123     value(2) = pi*(r2);
124     value(3) = pi*(r3);
125
126     figure(2)
127     hold on
```

```
128     p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'K.','MarkerSize', ...
            value(1)+fator,'LineWidth',2);
129     p2 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'K.','MarkerSize', ...
            value(2)+fator,'LineWidth',2);
130     p3 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'K.','MarkerSize', ...
            value(3)+fator,'LineWidth',2);
131     p4 = plot(x(1,:),y(1,:),'K','LineWidth',2)
132     daspect([1 1 1])
133     hold off
134     axis([-1.5 1.5 -1.5 1.5])
135     Pts_aprox
136     alfa_aprox
137 end
138 figure(3)
139 hold on
140 p1 = plot(xygr(:,1),xygr(:,2),'k.','LineWidth',2);
141 axis([-1.5 1.5 -1.5 1.5])
142 p2 = plot(x(1,:),y(1,:),'K','LineWidth',2);
143 daspect([1 1 1])
144 hold off
145 toc
146 lambda = +1.0;
147 R=3.0;
148 r=1.0;
149 %Calcula a integral de KV com b_0=b*
150 J = integralKV(m,alfa_aprox,Pts_aprox,u_star,q_star,M,N,R,r,lambda)
```

```
1  %EXEMPLO 7
2  clear all
3  close all
4  clc
5
6  tic
7  %Tamanho do grid
8  L_grid = 100;
9  %intensidade das cargas
10 alpha(1,1) = 5;
11 alpha(2,1) = 10;
12 alpha(3,1) = 15;
13
14 % alpha(1,1) = 5;
15 % alpha(2,1) = 5;
16 % alpha(3,1) = 5;
17
18 %Grid gerado com sunflower seed
19 [p1 p2] = sunflower_2D(L_grid,0);
```

```
20 p = [p1 p2];
21 %Lambda 1
22 pto1=63;
23 pto2=22;
24 pto3=52;
25
26 %Lambda 2
27 % pto1=27;
28 % pto2=47;
29 % pto3=67;
30
31
32
33 %Localizacao das cargas
34 xyst(1,1) =   p(pto1,1);
35 xyst(1,2) =   p(pto1,2);
36
37 xyst(2,1) =   p(pto2,1);
38 xyst(2,2) =   p(pto2,2);
39
40 xyst(3,1) =   p(pto3,1);
41 xyst(3,2) =   p(pto3,2);
42
43 % figure(1)
44 % plot(xyst(:,1),xyst(:,2),'k.','MarkerSize', 35,'LineWidth',2)
45 % hold on
46 % [x,y] = cylinder(1,1000);
47 % plot(x(1,:),y(1,:),'K','LineWidth',2)
48 % daspect([1 1 1])
49 % axis([-1.5 1.5 -1.5 1.5])
50 % hold off
51 % STOP
52
53 %ruido (em porcentagem)
54 noise = 40;
55 %variavel auxiliar
56 fator = 30;
57
58 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
      mainHelm_2D(alpha, xyst, noise, L_grid);
59 alfa_aprox1 = alfa_aprox;
60 Pts_aprox1 = Pts_aprox;
61 M = size(q_star,1);
62 z = linspace(0,1,M);
63 [x,y] = cylinder(1,1000);
64
65 r1 = alfa_aprox(1,1)/2;
```

```
r2 = alfa_aprox(2,1)/2;
r3 = alfa_aprox(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(2)
hold on
p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
    value(1) + fator,'LineWidth',2);
p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
    value(2) + fator ,'LineWidth',2);
p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
    value(3)+ fator ,'LineWidth',2);

p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
daspect([1 1 1])
hold off
axis([-1.5 1.5 -1.5 1.5])
figure(3)
plot(z,q_star,'k.-',z,q_star_noise,'k*');
title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
    ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
    ', ', num2str(alfa_aprox(3,1))])
legend('without noise','with noise')
daspect([1 1 1])

abs(q_star - q_star_noise)./q_star
asaasdasd

noise = 5;
[alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
    mainHelm_2D(alpha, xyst, noise, L_grid);
alfa_aprox2 = alfa_aprox;
Pts_aprox2 = Pts_aprox;
r1 = alfa_aprox(1,1)/2;
r2 = alfa_aprox(2,1)/2;
r3 = alfa_aprox(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(4)
hold on
p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
    value(1) + fator,'LineWidth',2);
p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
```

```
      value(2) + fator ,'LineWidth',2);
106 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
      value(3)+ fator ,'LineWidth',2);
107
108 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
109 daspect([1 1 1])
110 hold off
111 axis([-1.5 1.5 -1.5 1.5])
112
113 figure(5)
114 plot(z,q_star,'k.-',z,q_star_noise,'k*');
115 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
      ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
      ', ', num2str(alfa_aprox(3,1))])
116 legend('without noise','with noise')
117 daspect([1 1 1])
118
119 noise = 10;
120 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
      mainHelm_2D(alpha, xyst, noise, L_grid);
121 alfa_aprox3 = alfa_aprox;
122 Pts_aprox3 = Pts_aprox;
123 r1 = alfa_aprox(1,1)/2;
124 r2 = alfa_aprox(2,1)/2;
125 r3 = alfa_aprox(3,1)/2;
126
127 value(1) = pi*(r1);
128 value(2) = pi*(r2);
129 value(3) = pi*(r3);
130 figure(6)
131 hold on
132 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
      value(1) + fator,'LineWidth',2);
133 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
      value(2) + fator ,'LineWidth',2);
134 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
      value(3)+ fator ,'LineWidth',2);
135
136 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
137 daspect([1 1 1])
138 hold off
139 axis([-1.5 1.5 -1.5 1.5])
140
141 figure(7)
142 plot(z,q_star,'k.-',z,q_star_noise,'k*');
143 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
      ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
```

```
      ', ', num2str(alfa_aprox(3,1))])
144 legend('without noise','with noise')
145 daspect([1 1 1])
146
147 noise = 20;
148 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
      mainHelm_2D(alpha, xyst, noise, L_grid);
149 alfa_aprox4 = alfa_aprox;
150 Pts_aprox4 = Pts_aprox;
151 r1 = alfa_aprox(1,1)/2;
152 r2 = alfa_aprox(2,1)/2;
153 r3 = alfa_aprox(3,1)/2;
154
155 value(1) = pi*(r1);
156 value(2) = pi*(r2);
157 value(3) = pi*(r3);
158 figure(8)
159 hold on
160 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
      value(1) + fator,'LineWidth',2);
161 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
      value(2) + fator ,'LineWidth',2);
162 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
      value(3)+ fator ,'LineWidth',2);
163
164 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
165 daspect([1 1 1])
166 hold off
167 axis([-1.5 1.5 -1.5 1.5])
168
169 figure(9)
170 plot(z,q_star,'k.-',z,q_star_noise,'k*');
171 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
      ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
      ', ', num2str(alfa_aprox(3,1))])
172 legend('without noise','with noise')
173 daspect([1 1 1])
174
175 noise = 40;
176 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
      mainHelm_2D(alpha, xyst, noise, L_grid);
177 alfa_aprox5 = alfa_aprox;
178 Pts_aprox5 = Pts_aprox;
179 r1 = alfa_aprox(1,1)/2;
180 r2 = alfa_aprox(2,1)/2;
181 r3 = alfa_aprox(3,1)/2;
182
```

```
value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(10)
hold on
p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
    value(1) + fator,'LineWidth',2);
p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
    value(2) + fator ,'LineWidth',2);
p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
    value(3)+ fator ,'LineWidth',2);

p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
daspect([1 1 1])
hold off
axis([-1.5 1.5 -1.5 1.5])

figure(11)
plot(z,q_star,'k.-',z,q_star_noise,'k*');
title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
    ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
    ', ', num2str(alfa_aprox(3,1))])
legend('without noise','with noise')
daspect([1 1 1])
```

```
%EXEMPLO 12
clear all
close all
clc

tic
%Tamanho do grid
L_grid = 100;
%Intensidade das cargas
alpha(1,1) = 6;
alpha(2,1) = 14;
alpha(3,1) = 17;

%Grid gerado com sunflower seed
[p1 p2] = sunflower_2D(L_grid,0);
p = [p1 p2];
pto1=2;
pto2=48;
pto3=76;

xyst(1,1) =   p(pto1,1);
```

```
xyst(1,2) =   p(pto1,2);

xyst(2,1) =   p(pto2,1);
xyst(2,2) =   p(pto2,2);

xyst(3,1) =   p(pto3,1);
xyst(3,2) =   p(pto3,2);

% figure(1)
% plot(xyst(:,1),xyst(:,2),'k.','MarkerSize', 35,'LineWidth',2)
% hold on
% [x,y] = cylinder(1,1000);
% plot(x(1,:),y(1,:),'K','LineWidth',2)
% daspect([1 1 1])
% axis([-1.5 1.5 -1.5 1.5])
% hold off
% STOP

%ruido (em porcentagem)
noise = 0;
fator = 30;

[alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
    mainHelm_2D(alpha, xyst, noise, L_grid);
alfa_aprox1 = alfa_aprox;
Pts_aprox1 = Pts_aprox;
M = size(q_star,1);
z = linspace(0,1,M);
[x,y] = cylinder(1,1000);

r1 = alfa_aprox(1,1)/2;
r2 = alfa_aprox(2,1)/2;
r3 = alfa_aprox(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(2)
hold on
p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
    value(1) + fator,'LineWidth',2);
p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
    value(2) + fator ,'LineWidth',2);
p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
    value(3)+ fator ,'LineWidth',2);

p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
```

```
65 daspect([1 1 1])
66 hold off
67 axis([-1.5 1.5 -1.5 1.5])
68
69 figure(3)
70 plot(z,q_star,'k.-',z,q_star_noise,'k*');
71 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
72 legend('without noise','with noise')
73 hjkhjkhjk
74 noise = 5;
75 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainHelm_2D(alpha, xyst, noise, L_grid);
76 alfa_aprox2 = alfa_aprox;
77 Pts_aprox2 = Pts_aprox;
78 r1 = alfa_aprox(1,1)/2;
79 r2 = alfa_aprox(2,1)/2;
80 r3 = alfa_aprox(3,1)/2;
81
82 value(1) = pi*(r1);
83 value(2) = pi*(r2);
84 value(3) = pi*(r3);
85
86 figure(4)
87 hold on
88 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
       value(1) + fator,'LineWidth',2);
89 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
       value(2) + fator ,'LineWidth',2);
90 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
       value(3)+ fator ,'LineWidth',2);
91 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
92 daspect([1 1 1])
93 hold off
94 axis([-1.5 1.5 -1.5 1.5])
95
96 figure(5)
97 plot(z,q_star,'k.-',z,q_star_noise,'k*');
98 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
99 legend('without noise','with noise')
100 noise = 10;
101 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainHelm_2D(alpha, xyst, noise, L_grid);
102 alfa_aprox3 = alfa_aprox;
```

```
103 Pts_aprox3 = Pts_aprox;
104 r1 = alfa_aprox(1,1)/2;
105 r2 = alfa_aprox(2,1)/2;
106 r3 = alfa_aprox(3,1)/2;
107
108 value(1) = pi*(r1);
109 value(2) = pi*(r2);
110 value(3) = pi*(r3);
111 figure(6)
112 hold on
113 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
       value(1) + fator,'LineWidth',2);
114 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
       value(2) + fator ,'LineWidth',2);
115 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
       value(3)+ fator ,'LineWidth',2);
116
117 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
118 daspect([1 1 1])
119 hold off
120 axis([-1.5 1.5 -1.5 1.5])
121
122 figure(7)
123 plot(z,q_star,'k.-',z,q_star_noise,'k*');
124 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
125 legend('without noise','with noise')
126
127 noise = 20;
128 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainHelm_2D(alpha, xyst, noise, L_grid);
129 alfa_aprox4 = alfa_aprox;
130 Pts_aprox4 = Pts_aprox;
131 r1 = alfa_aprox(1,1)/2;
132 r2 = alfa_aprox(2,1)/2;
133 r3 = alfa_aprox(3,1)/2;
134
135 value(1) = pi*(r1);
136 value(2) = pi*(r2);
137 value(3) = pi*(r3);
138 figure(8)
139 hold on
140 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
       value(1) + fator,'LineWidth',2);
141 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
       value(2) + fator ,'LineWidth',2);
```

```
142 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
       value(3)+ fator ,'LineWidth',2);
143
144 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
145 daspect([1 1 1])
146 hold off
147 axis([-1.5 1.5 -1.5 1.5])
148
149 figure(9)
150 plot(z,q_star,'k.-',z,q_star_noise,'k*');
151 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
152 legend('without noise','with noise')
153
154 noise = 40;
155 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
       mainHelm_2D(alpha, xyst, noise, L_grid);
156 alfa_aprox5 = alfa_aprox;
157 Pts_aprox5 = Pts_aprox;
158 r1 = alfa_aprox(1,1)/2;
159 r2 = alfa_aprox(2,1)/2;
160 r3 = alfa_aprox(3,1)/2;
161
162 value(1) = pi*(r1);
163 value(2) = pi*(r2);
164 value(3) = pi*(r3);
165 figure(10)
166 hold on
167 p1 = plot(Pts_aprox(1,1),Pts_aprox(1,2),'k.','MarkerSize', ...
       value(1) + fator,'LineWidth',2);
168 p3 = plot(Pts_aprox(2,1),Pts_aprox(2,2),'k.','MarkerSize', ...
       value(2) + fator ,'LineWidth',2);
169 p5 = plot(Pts_aprox(3,1),Pts_aprox(3,2),'k.','MarkerSize', ...
       value(3)+ fator ,'LineWidth',2);
170
171 p7 = plot(x(1,:),y(1,:),'K','LineWidth',2)
172 daspect([1 1 1])
173 hold off
174 axis([-1.5 1.5 -1.5 1.5])
175
176 figure(11)
177 plot(z,q_star,'k.-',z,q_star_noise,'k*');
178 title(['Noise: ',num2str(noise),'%',' -  Aprox. alpha: ...
       ',num2str(alfa_aprox(1,1)), ', ',num2str(alfa_aprox(2,1)), ...
       ', ', num2str(alfa_aprox(3,1))])
179 legend('without noise','with noise')
```

Equação do Tipo Helmholtz Tridimensional

Programa Principal

```
%Ex. 8
% function [valor_alfa, Pts_otimo, xyzc, xyzs, xyzgr] = ...
    mainHelm_3D(alpha, xyzst, L_Grid,lambda,R)

%Ex. 9
% function [valor_alfa, Pts_otimo, xyzc, xyzs, xyzgr] = ...
    mainHelm_3D(alpha, xyzst, L_Grid)

%Ex. 10
function [valor_alfa, Pts_otimo, xyzc, xyzs, xyzgr] = ...
    mainHelm_3D(alpha, xyzst,L_Grid,m)

%Ex. 11 (Ex. 13)
%  function [valor_alfa, Pts_otimo, q_star_without_noise, ...
    q_star_with_noise, xyzc, xyzs, xyzgr] = mainHelm_3D(alpha, ...
    xyzst, noise, L_Grid)

% lambda = + 9.5;
% lambda = + 1;
lambda = - 4;

%tolerancia para a funcao lsqr
tol=1e-15;

%raio do dominio
r = 1.0;

%raio do fronteira ficticia
%R = 3.5;
R = 2.0;

%Numero de pontos de colocacao
n_1 = 3;

%Numero total de pontos de colocacao
M = sphere_cubed_point_num ( n_1 );

xyzc = sphere_cubed_points ( n_1, M );
xyzc = xyzc';

%Vetor normal
nxyz = xyzc;
```

```
38
39 xyzc(:,1) = r*xyzc(:,1);
40 xyzc(:,2) = r*xyzc(:,2);
41 xyzc(:,3) = r*xyzc(:,3);
42
43 %Pontos fonte
44 n_2 = 3;
45
46 %Numero total de pontos fonte
47 N = sphere_cubed_point_num ( n_2 );
48
49 xyzs = sphere_cubed_points ( n_2, N );
50 xyzs = xyzs';
51 xyzs(:,1) = R*xyzs(:,1);
52 xyzs(:,2) = R*xyzs(:,2);
53 xyzs(:,3) = R*xyzs(:,3);
54
55 %Numero de cargas para serem reconstruidas
56 numPtsfontcurr = size(alpha,1);
57
58 %Numero de pontos para integracao de superficie (valores ...
      tabelados, vide getLebedevSphere)
59 %Angular grid unrecognized, choices are 6, 14, 26, 38, 50, 74, ...
      86, 110, 146, 170, 194, 230, 266, 302, 350, 434,
60 %590, 770, 974, 1202, 1454, 1730, 2030, 2354, 2702, 3074, 3470, ...
      3890, 4334, 4802, 5294, 5810
61 MT = 434;
62
63 %Pontos na fronteira fisica para integracao numerica
64 res = getLebedevSphere(MT);
65 x_int = res.x;
66 y_int = res.y;
67 z_int = res.z;
68 pesos_int = res.w;
69
70 %compacta em uma unica variavel
71 xyz_int = [ x_int y_int z_int ];
72
73 % Gera artificalmente os dados de Cauchy
74 [q_star, u_star] = calc_q_star_hel_3D(xyzs,xyzc,nxyz,alpha,...
75                     xyzst,numPtsfontcurr,lambda);
76
77 % %acrescenta ruido
78 % q_star_without_noise = q_star;
79 % q_star_with_noise = q_star.*(1+ noise*(2*rand(M,1)-1)/100 );
80 % q_star = q_star_with_noise;
81
```

```
%%
%calcula os coeficientes de uD
M_uD=[];
for i=1:M
    for j=1:N
        raio2 = (xyzc(i,1)-xyzs(j,1))^2 + ...
            (xyzc(i,2)-xyzs(j,2))^2 + (xyzc(i,3)-xyzs(j,3))^2;
        if (lambda < 0)
            arg = 1i*sqrt(-lambda)*sqrt(raio2);
        else
            arg = sqrt(lambda)*sqrt(raio2);
        end
        M_uD(i,j) = real( exp(arg) )/(4*pi*sqrt(raio2));
    end
    vetor_uD(i,1) = u_star(i);
end

%coeficientes de uD
c_uD = lsqr(M_uD,vetor_uD,tol);

%calcula os coeficientes de uN
M_uN=[];
for i=1:M
    for j=1:N
        raio2 = (xyzc(i,1)-xyzs(j,1))^2 + ...
            (xyzc(i,2)-xyzs(j,2))^2 + (xyzc(i,3)-xyzs(j,3))^2;
        if (lambda < 0)
            arg = 1i*sqrt(-lambda)*sqrt(raio2);
        else
            arg = sqrt(lambda)*sqrt(raio2);
        end
        %Derivada normal de uN
        DuN_x(i,j) = real( ( (xyzc(i,1)-xyzs(j,1))*exp(arg)*(arg ...
            -1) )/( 4*pi*sqrt(raio2)*raio2 ) );
        DuN_y(i,j) = real( ( (xyzc(i,2)-xyzs(j,2))*exp(arg)*(arg ...
            -1) )/( 4*pi*sqrt(raio2)*raio2 ) );
        DuN_z(i,j) = real( ( (xyzc(i,3)-xyzs(j,3))*exp(arg)*(arg ...
            -1) )/( 4*pi*sqrt(raio2)*raio2 ) );
        M_uN(i,j)  = DuN_x(i,j)*nxyz(i,1) + DuN_y(i,j)*nxyz(i,2) ...
            + DuN_z(i,j)*nxyz(i,3);
    end
    vetor_uN(i,1) = - q_star(i);
end

%Coeficientes de uN
c_uN = lsqr(M_uN,vetor_uN,tol);
%%
```

```
123 %calcula uD e uN na fronteira para integracao numerica
124 matriz_uD_uN = [];
125 for i=1:MT
126     for j=1:N
127         raio2 = (xyz_int(i,1)-xyzs(j,1))^2 + ...
                (xyz_int(i,2)-xyzs(j,2))^2 + (xyz_int(i,3)-xyzs(j,3))^2;
128         if (lambda < 0)
129             arg = 1i*sqrt(-lambda)*sqrt(raio2);
130         else
131             arg = sqrt(lambda)*sqrt(raio2);
132         end
133         matriz_uD_uN(i,j) = real( exp(arg) )/(4*pi*sqrt(raio2));
134     end
135 end
136 uD_int = matriz_uD_uN*c_uD;
137 uN_int = matriz_uD_uN*c_uN;
138 %%
139
140 %Grid de pontos desenvolvido por John Burkardt, (distributed ...
       under the GNU LGPL license.)
141 tam = ball_grid_count ( L_Grid, r, centro );
142 bg = ball_grid ( L_Grid, r, centro, tam );
143
144 %adaptacao para a variavel do programa
145 for i=1:tam
146     xyzgr(i,:) = bg(1:3,i);
147 end
148
149 %hi: valor aproximado de hi nos pontos da integracao de superficie
150 %di: valor aproximado de hi(uD-uN) nos pontos da integracao de ...
       superficie
151 [hi di] = calc_hi_di_hel_3D(...
152                     xyzc,nxyz,xyzs,xyz_int,...
153                     xyzgr,uD_int,uN_int,...
154                     tol,...
155                     lambda,tam);
156 aux = 1;
157 numPtsfontcurr = m;
158
159 switch numPtsfontcurr
160     case 1
161         for i1 = 1:tam
162             entry_hi = hi(:,i1);
163             entry_di = di(:,i1);
164             [H d] = calc_H_d_hel_3D(entry_hi, ...
                    entry_di,numPtsfontcurr,pesos_int);
165             alfas(:,aux) = lsqr(H,d,tol);
```

```
166             funcional(aux,1)=-0.5*alfas(:,aux)'*d;
167             aux_index(:,aux) = i1;
168             aux = aux + 1;
169         end
170     case 2
171         for i1 = 1:tam
172             for i2 = i1:tam
173                 entry_hi = [ hi(:,i2) hi(:,i1) ];
174                 entry_di = [ di(:,i2) di(:,i1) ];
175                 [H d] = calc_H_d_hel_3D(entry_hi, ...
                        entry_di,numPtsfontcurr,pesos_int);
176                 alfas(:,aux) = lsqr(H,d,tol);
177                 funcional(aux,1)=-0.5*alfas(:,aux)'*d;
178                 aux_index(:,aux) = [i2 i1];
179                 aux = aux + 1;
180             end
181         end
182     case 3
183         for i1 = 1:tam
184             for i2 = i1:tam
185                 for i3 = i2:tam
186                     entry_hi = [ hi(:,i3) hi(:,i2) hi(:,i1)];
187                     entry_di = [ di(:,i3) di(:,i2) di(:,i1)];
188                     [H d] = calc_H_d_hel_3D(entry_hi, ...
                            entry_di,numPtsfontcurr,pesos_int);
189                     alfas(:,aux) = lsqr(H,d,tol);
190                     funcional(aux,1)=-0.5*alfas(:,aux)'*d;
191                     aux_index(:,aux) = [i3 i2 i1];
192                     aux = aux + 1;
193                 end
194             end
195         end
196     case 4
197         for i1 = 1:tam
198             for i2 = i1:tam
199                 for i3 = i2:tam
200                     for i4 = i3:tam
201                         entry_hi = [ hi(:,i3) hi(:,i2) hi(:,i1) ...
                                hi(:,i4)];
202                         entry_di = [ di(:,i3) di(:,i2) di(:,i1) ...
                                di(:,i4)];
203                         [H d] = calc_H_d_hel_3D(entry_hi, ...
                                entry_di,numPtsfontcurr,pesos_int);
204                         alfas(:,aux) = lsqr(H,d,tol);
205                         funcional(aux,1)=-0.5*alfas(:,aux)'*d;
206                         aux_index(:,aux) = [i3 i2 i1 i4];
207                         aux = aux + 1;
```

```
208                     end
209                 end
210             end
211         end
212     otherwise
213 end
214
215 [value, index] = min(funcional);
216 funcional  = value;
217 valor_alfa = alfas(:,index);
218 Pts_otimo = xyzgr(aux_index(:,index),:);
```

Rotinas Utilizadas

```
1  function [q_star, u_star]=calc_q_star_hel_3D(xyzs,xyzc,...
2                               nxyz,alpha,xyzst,numPtsfontcurr,lambda)
3  %Numero de pontos de colocacao
4  M = size(xyzc,1);
5  %Numero de pontos de fonte
6  N = size(xyzs,1);
7  %Dado de Dirichlet prescrito
8  u_star = zeros(M,1);
9  %Calcula a derivada normal e a matriz do sistema
10 for i = 1:M
11     %Parcela da solucao aproximada associada ao somatorio
12     for j = 1:N
13         M_qst_H(i,j) = (xyzc(i,1) - xyzs(j,1))^2 + (xyzc(i,2) - ...
               xyzs(j,2))^2 + (xyzc(i,3) - xyzs(j,3))^2;
14         if (lambda < 0)
15             arg = 1i*sqrt(-lambda)*sqrt(M_qst_H(i,j));
16         else
17             arg = sqrt(lambda)*sqrt(M_qst_H(i,j));
18         end
19         %Derivada normal da solucao fundamental
20         Gx_H(i,j) =  real( ( (xyzc(i,1) - ...
               xyzs(j,1))*exp(arg)*(arg -1) )/( ...
               4*pi*sqrt(M_qst_H(i,j))*M_qst_H(i,j) ) );
21         Gy_H(i,j) = real( ( (xyzc(i,2) - ...
               xyzs(j,2))*exp(arg)*(arg -1) )/( ...
               4*pi*sqrt(M_qst_H(i,j))*M_qst_H(i,j) ) );
22         Gz_H(i,j) = real( ( (xyzc(i,3) - ...
               xyzs(j,3))*exp(arg)*(arg -1) )/( ...
               4*pi*sqrt(M_qst_H(i,j))*M_qst_H(i,j) ) );
23         AN(i,j) = Gx_H(i,j)*nxyz(i,1) + Gy_H(i,j)*nxyz(i,2) + ...
               Gz_H(i,j)*nxyz(i,3);
24         %matriz do sistema
```

```
25          AD(i,j) = real( exp(arg))/(4*pi*sqrt(M_qst_H(i,j)));
26      end
27      fun(i,1)= u_star(i);
28      q_star2(i,1) = 0;
29      for k=1:numPtsfontcurr
30          M_qst(i,1,k) = (xyzc(i,1) - xyzst(k,1))^2 + (xyzc(i,2) - ...
                xyzst(k,2))^2 + (xyzc(i,3) - xyzst(k,3))^2;
31          if (lambda < 0)
32              arg = 1i*sqrt(-lambda)*sqrt(M_qst(i,1,k));
33          else
34              arg = sqrt(lambda)*sqrt(M_qst(i,1,k));
35          end
36          fun(i,1) = fun(i,1) - alpha(k,1)*real( exp(arg) ...
                )/(4*pi*sqrt(M_qst(i,1,k)));
37          %parcela da solucao aprox associada a alpha_i
38          Gx(i,1,k) =  real( ( (xyzc(i,1) - ...
                xyzst(k,1))*exp(arg)*(arg -1) )/( ...
                4*pi*sqrt(M_qst(i,1,k))*M_qst(i,1,k) ) );
39          Gy(i,1,k) =  real( ( (xyzc(i,2) - ...
                xyzst(k,2))*exp(arg)*(arg -1) )/( ...
                4*pi*sqrt(M_qst(i,1,k))*M_qst(i,1,k) ) );
40          Gz(i,1,k) =  real( ( (xyzc(i,3) - ...
                xyzst(k,3))*exp(arg)*(arg -1) )/( ...
                4*pi*sqrt(M_qst(i,1,k))*M_qst(i,1,k) ) );
41          q_star2(i,1) = q_star2(i,1) + alpha(k,1)*( ...
                Gx(i,1,k)*nxyz(i,1) + Gy(i,1,k)*nxyz(i,2) + ...
                Gz(i,1,k)*nxyz(i,3) );
42      end
43  end
44  cD = lsqr(AD,fun);
45  q_star1=AN*cD;
46  q_star = -( q_star1 + q_star2 );
```

```
1   function [hi di] = calc_hi_di_hel_3D(...
2                               xyzc,nxyz,xyzs,xyz_int,...
3                               pts_gr,u_star_integral,uN,...
4                               tol,...
5                               lambda,numPts)
6   %%
7   %Numero de pontos de colocacao
8   M = size(xyzc,1);
9   %Numero de pontos fonte
10  N = size(xyzs,1);
11  %Numero de pontos para a integracao numerica
12  MT = size(xyz_int,1);
13  %Matriz utilizada para calcular os coeficientes de vi
```

```
matriz_vi = [];
for i=1:M
    for j=1:N
        raio2 = (xyzc(i,1) - xyzs(j,1))^2 + (xyzc(i,2) - ...
            xyzs(j,2))^2 + (xyzc(i,3) - xyzs(j,3))^2;
        if(lambda < 0)
            arg = 1i*sqrt(-lambda)*sqrt(raio2);
        else
            arg = sqrt(lambda)*sqrt(raio2);
        end
        matriz_vi(i,j) = real( exp(arg) )/(4*pi*sqrt(raio2));
        %Matriz utilizada para calcular a derivada normal de vi
        %Tambem sera utilizada para a matriz no problema hi
        matriz_Dvi1_hiH_x(i,j) = real( ( (xyzc(i,1) - ...
            xyzs(j,1))*exp(arg)*(arg-1) )/( ...
            4*pi*sqrt(raio2)*raio2 ) );
        matriz_Dvi1_hiH_y(i,j) = real( ( (xyzc(i,2) - ...
            xyzs(j,2))*exp(arg)*(arg-1) )/( ...
            4*pi*sqrt(raio2)*raio2 ) );
        matriz_Dvi1_hiH_z(i,j) = real( ( (xyzc(i,3) - ...
            xyzs(j,3))*exp(arg)*(arg-1) )/( ...
            4*pi*sqrt(raio2)*raio2 ) );
        matriz_Dvi1_hiH(i,j)   = ...
            matriz_Dvi1_hiH_x(i,j)*nxyz(i,1) + ...
            matriz_Dvi1_hiH_y(i,j)*nxyz(i,2) + ...
            matriz_Dvi1_hiH_z(i,j)*nxyz(i,3);
    end
    %calcula a derivada normal da solucao particular de vi
    for k = 1:numPts
        dist_vet(i,1,k) = (xyzc(i,1) - pts_gr(k,1))^2 + ...
            (xyzc(i,2) - pts_gr(k,2))^2 + (xyzc(i,3) - ...
            pts_gr(k,3))^2;
        if(lambda < 0)
            arg = 1i*sqrt(-lambda)*sqrt(dist_vet(i,1,k));
        else
            arg = sqrt(lambda)*sqrt(dist_vet(i,1,k));
        end
        %vetor utiliado para calcular os coeficientes de vi
        vetor_vi(i,1,k) = - real( exp(arg) ...
            )/(4*pi*sqrt(dist_vet(i,1,k)));

        Dvi2_x(i,1,k) =  real( ( (xyzc(i,1) - ...
            pts_gr(k,1))*exp(arg)*(arg-1) )/( ...
            4*pi*sqrt(dist_vet(i,1,k))*dist_vet(i,1,k) ) );
        Dvi2_y(i,1,k) =  real( ( (xyzc(i,2) - ...
            pts_gr(k,2))*exp(arg)*(arg-1) )/( ...
            4*pi*sqrt(dist_vet(i,1,k))*dist_vet(i,1,k) ) );
```

```
44          Dvi2_z(i,1,k) =  real( ( (xyzc(i,3) - ...
                pts_gr(k,3))*exp(arg)*(arg-1) )/( ...
                4*pi*sqrt(dist_vet(i,1,k))*dist_vet(i,1,k) ) );
45          Dvi2(i,1,k)    = Dvi2_x(i,1,k)*nxyz(i,1) + ...
                Dvi2_y(i,1,k)*nxyz(i,2) + Dvi2_z(i,1,k)*nxyz(i,3);
46      end
47  end
48  %coeficientes de vi
49  for k=1:numPts
50      cvi(:,k) = lsqr(matriz_vi,vetor_vi(:,1,k),tol);
51      Dvi1(:,k) = matriz_Dvi1_hiH*cvi(:,k);
52      Dvi(:,k) = Dvi1(:,k) + Dvi2(:,1,k);
53      vetor_hiH(:,1,k) = - Dvi(:,k);
54  end
55  %coeficientes de hi
56  for k=1:numPts
57      chi(:,k) = lsqr(matriz_Dvi1_hiH,vetor_hiH(:,1,k),tol);
58  end
59  %%
60  %calcula hi e di
61  for i=1:MT
62      for j=1:N
63          raio2 = sqrt( (xyz_int(i,1) - xyzs(j,1))^2 + ...
                (xyz_int(i,2) - xyzs(j,2))^2 + (xyz_int(i,3) - ...
                xyzs(j,3))^2 );
64          if (lambda < 0)
65              arg = 1i*sqrt(-lambda)*raio2;
66          else
67              arg = sqrt(lambda)*raio2;
68          end
69          matriz_hiH(i,j) = real( exp(arg) )/(4*pi*raio2);
70      end
71      for k=1:numPts
72          hi(i,k) = matriz_hiH(i,:)*chi(:,k);
73          di(i,k) = hi(i,k)*(u_star_integral(i) - uN(i));
74      end
75  end
```

```
1   function [H d] = calc_H_d_hel_3D(hi, di,numPtsfontcurr,pesos)
2   for i = 1:numPtsfontcurr
3       for j=i:numPtsfontcurr
4           if (i==j)
5               funHij = hi(:,i).^2;
6               H(i,j) = sum(funHij.*pesos);
7           else
8               funHij = hi(:,i).*hi(:,j);
```

```
            H(i,j) = sum(funHij.*pesos);
            H(j,i) = H(i,j);
        end
    end
    d(i,1) = sum(di(:,i).*pesos);
end
```

Exemplos Numéricos

```
%EXEMPLO 8
clear all
close all
clc

tic
%Intensidade da carga
alpha(1,1) = 2;
% alpha(1,1) = 4;

%Tamanho do grid
L_Grid = 2.5;
%Grid gerado com ball_grid_count
tam = ball_grid_count ( L_Grid, 1.0, [0,0,0] );
bg = ball_grid ( L_Grid, 1.0, [0,0,0], tam );
%Lambda_1
% pt1 = 97;

%Lambda_2
% pt1 = 50;

%Lambda_3
pt1 = 50;

p(1,:) = bg(1:3,pt1);
%Localizacao das cargas
dx_dy=0.0;
xyzst(1,1) =   p(1,1)+dx_dy;
xyzst(1,2) =   p(1,2)-dx_dy;
xyzst(1,3) =   p(1,3)+dx_dy;
% lambda = +9.5;
% lambda = +1;
lambda = -4;
R = 1.1 : 0.1 : 3;
L =  length(R)
aux=1;
for i=1:L
```

```
38      [alfa_aprox, Pts_aprox, xyzc, xyzs, xyzgr] = ...
            mainHelm_3D(alpha, xyzst, L_Grid, lambda, R(i));
39      erroABS_alpha(aux) = abs(alpha - alfa_aprox);
40      erroREL_alpha(aux) = erroABS_alpha(aux) / abs(alpha);
41      distEUC(aux) = norm(Pts_aprox - xyzst);
42      aux = aux+1;
43  end
44  figure(1)
45  plot(R, 100*erroREL_alpha,'b.-','MarkerSize', ...
        10,'LineWidth',1,'LineStyle','--')
46  xlabel('Raio R') % x-axis label
47  ylabel('Erro Relativo de \alpha (%)') % y-axis label
48  title('Curva de Erro')
49
50  figure(2)
51  plot(R, distEUC,'b.-','MarkerSize', ...
        10,'LineWidth',1,'LineStyle','--')
52  xlabel('Raio R') % x-axis label
53  ylabel('Norma Euclidiana |x - x^*|') % y-axis label
54  toc
```

```
1   %EXEMPLO 9
2   clear all
3   close all
4   clc
5
6   tic
7   %Intensidade da carga
8   alpha(1,1) = 15;
9
10  %Tamanho do grid
11  % L_Grid = 1.3 %(27 pontos)
12  L_Grid = 2.4 %(93 pontos)
13  % L_Grid = 4.1 %(437 pontos)
14  % L_Grid = 5.3 %(799 pontos)
15  % L_Grid = 7.99 %(2403 pontos)
16
17  %Grid gerado com ball_grid
18  tam = ball_grid_count ( L_Grid, 1.0, [0,0,0] );
19  bg = ball_grid ( L_Grid, 1.0, [0,0,0], tam );
20  %pt1 = 50;
21  pt1 = 72;
22  p(1,:) = bg(1:3,pt1)
23
24  %L_Grid=10: 4945 pts
25  %L_Grid=9 : 3695 pts
```

```
%L_Grid=8 : 2553 pts
%L_Grid=7 : 1791 pts
%L_Grid=6 : 1189 pts
%L_Grid=5 : 739 pts
%L_Grid=4 : 389 pts
%L_Grid=3 : 179 pts
%L_Grid=2 : 81 pts

%Localizacao das cargas
dx_dy=0.0;
xyzst(1,1) =   p(1,1)+dx_dy;
xyzst(1,2) =   p(1,2)-dx_dy;
xyzst(1,3) =   p(1,3)+dx_dy;

%Localizacao das cargas
% xyzst(1,1) =  -0.25;
% xyzst(1,2) =  0;
% xyzst(1,3) =  0.35;

% xyzst(1,1) =   -0.25;
% xyzst(1,2) =   0.34;
% xyzst(1,3) =  0.21;
%solucao aproximada
[alfa_aprox, Pts_aprox, xyzc, xyzs, xyzgr] = mainHelm_3D(alpha, ...
    xyzst, L_Grid);
figure(1)
p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),...
                Pts_aprox(1,3),'r.','MarkerSize', 40,'LineWidth',2);
hold on
p2 = plot3(xyzst(1,1),xyzst(1,2),xyzst(1,3),'g.','MarkerSize', ...
    40,'LineWidth',2);

[x y z] = sphere(50);
h = surfl(x, y, z);
set(h, 'FaceAlpha', 0.1)
daspect([1 1 1])
hold off
title(['Computed alpha: ',num2str(alfa_aprox(1,1))])
toc
alfa_aprox
Pts_aprox
ball_grid_count ( L_Grid, 1.0, [0,0,0] )
erro_loc = norm(xyzst - Pts_aprox)
erro_int = abs(alpha - alfa_aprox)/abs(alpha)
figure(2)
plot3(xyzgr(:,1),xyzgr(:,2),xyzgr(:,3),'r.','MarkerSize', ...
    10,'LineWidth',2);
```

```
70 hold on
71 h = surfl(x, y, z);
72 set(h, 'FaceAlpha', 0)
73 daspect([1 1 1])
74 hold off
75 toc
```

```
1  %EXEMPLO 10
2  clear all
3  close all
4  clc
5
6  tic
7  %Intensidade das cargas
8  alpha(1,1) = 8;
9  alpha(2,1) = 8;
10 alpha(3,1) = 8;
11
12 % alpha(1,1) = 3;
13 % alpha(2,1) = 8;
14 % alpha(3,1) = 13;
15
16 %Numero de pontos do grid
17 L_Grid = 2.4;
18
19 %Grid gerado com ball_grid
20 tam = ball_grid_count ( L_Grid, 1.0, [0,0,0] );
21 bg = ball_grid ( L_Grid, 1.0, [0,0,0], tam );
22 pt1 = 8;
23 pt2 = 31;
24 pt3 = 85;
25
26 p(1,:) = bg(1:3,pt1);
27 p(2,:) = bg(1:3,pt2);
28 p(3,:) = bg(1:3,pt3);
29
30 %Localizacoes das cargas
31 xyzst(1,1) = p(1,1);
32 xyzst(1,2) = p(1,2);
33 xyzst(1,3) = p(1,3);
34
35 xyzst(2,1) = p(2,1);
36 xyzst(2,2) = p(2,2);
37 xyzst(2,3) = p(2,3);
38
39 xyzst(3,1) = p(3,1);
```

```
xyzst(3,2) = p(3,2);
xyzst(3,3) = p(3,3);

%Numero de cargas a serem reconstruidas
m = 4;
%Variavel auxiliar
fator = 35;

[alfa_aprox, Pts_aprox, xyzc, xyzs, xyzgr] = mainHelm_3D(alpha, ...
    xyzst, L_Grid,m);
toc
r1 = alpha(1,1)/2;
r2 = alpha(2,1)/2;
r3 = alpha(3,1)/2;

value_ex(1) = pi*(r1);
value_ex(2) = pi*(r2);
value_ex(3) = pi*(r3);

figure(1)
p2 = plot3(xyzst(1,1),xyzst(1,2),xyzst(1,3),'K.','MarkerSize', ...
    value_ex(1)+fator,'LineWidth',2);
hold on
p3 = plot3(xyzst(2,1),xyzst(2,2),xyzst(2,3),'K.','MarkerSize', ...
    value_ex(2)+fator,'LineWidth',2);
p4 = plot3(xyzst(3,1),xyzst(3,2),xyzst(3,3),'K.','MarkerSize', ...
    value_ex(3)+fator,'LineWidth',2);

[x y z] = sphere(50);
h = surfl(x, y, z);
set(h, 'FaceAlpha', 0)
daspect([1 1 1])
axis off
hold off

if(m==1)
    r1 = alfa_aprox(1,1)/2;
    value(1) = pi*(r1);

    figure(2)
    p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),...
                    Pts_aprox(1,3),'K.','MarkerSize', ...
                        value(1)+fator,'LineWidth',2);
    hold on
    [x y z] = sphere(50);
    h = surfl(x, y, z);
    set(h, 'FaceAlpha', 0)
```

```
82      daspect([1 1 1])
83      axis off
84      hold off
85      Pts_aprox
86      alfa_aprox
87  end
88  if(m==2)
89      r1 = alfa_aprox(1,1)/2;
90      r2 = alfa_aprox(2,1)/2;
91
92      value(1) = pi*(r1);
93      value(2) = pi*(r2);
94
95      figure(2)
96      p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),...
97                          Pts_aprox(1,3),'K.','MarkerSize', ...
                               value(1)+fator,'LineWidth',2);
98      hold on
99      p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),...
100                         Pts_aprox(2,3),'K.','MarkerSize', ...
                               value(2)+fator,'LineWidth',2);
101     [x y z] = sphere(50);
102     h = surfl(x, y, z);
103     set(h, 'FaceAlpha', 0)
104     daspect([1 1 1])
105     axis off
106     hold off
107     Pts_aprox
108     alfa_aprox
109 end
110 if(m==3)
111     r1 = alfa_aprox(1,1)/2;
112     r2 = alfa_aprox(2,1)/2;
113     r3 = alfa_aprox(3,1)/2;
114
115     value(1) = pi*(r1);
116     value(2) = pi*(r2);
117     value(3) = pi*(r3);
118
119     figure(2)
120     p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
121         'K.','MarkerSize', value(1)+fator,'LineWidth',2);
122     hold on
123     p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
124         'K.','MarkerSize', value(2)+fator,'LineWidth',2);
125     p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
126         'K.','MarkerSize', value(3)+fator,'LineWidth',2);
```

```
127     [x y z] = sphere(50);
128     h = surfl(x, y, z);
129     set(h, 'FaceAlpha', 0)
130     daspect([1 1 1])
131     axis off
132     hold off
133     Pts_aprox
134     alfa_aprox
135 end
136 if(m==4)
137     r1 = alfa_aprox(1,1)/2;
138     r2 = alfa_aprox(2,1)/2;
139     r3 = alfa_aprox(3,1)/2;
140     r4 = alfa_aprox(4,1)/2;
141 
142     value(1) = pi*(r1);
143     value(2) = pi*(r2);
144     value(3) = pi*(r3);
145     value(4) = pi*(r4);
146     tol = 2;
147     if ( value(1) ≤tol )
148         figure(2)
149         p1 = plot3(Pts_aprox(4,1),Pts_aprox(4,2),Pts_aprox(4,3),...
150             'K.','MarkerSize', value(4)+fator,'LineWidth',2);
151         hold on
152         p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
153             'K.','MarkerSize', value(2)+fator,'LineWidth',2);
154         p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
155             'K.','MarkerSize', value(3)+fator,'LineWidth',2);
156         [x y z] = sphere(50);
157         h = surfl(x, y, z);
158         set(h, 'FaceAlpha', 0)
159         daspect([1 1 1])
160         axis off
161         hold off
162     elseif ( value(2) ≤tol )
163         figure(2)
164         p1 = plot3(Pts_aprox(4,1),Pts_aprox(4,2),Pts_aprox(4,3),...
165             'K.','MarkerSize', value(4)+fator,'LineWidth',2);
166         hold on
167         p2 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
168             'K.','MarkerSize', value(1)+fator,'LineWidth',2);
169         p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
170             'K.','MarkerSize', value(3)+fator,'LineWidth',2);
171         [x y z] = sphere(50);
172         h = surfl(x, y, z);
173         set(h, 'FaceAlpha', 0)
```

```
174         daspect([1 1 1])
175         axis off
176         hold off
177     elseif ( value(3) ≤tol )
178         figure(2)
179         p1 = plot3(Pts_aprox(4,1),Pts_aprox(4,2),Pts_aprox(4,3),...
180             'K.','MarkerSize', value(4)+fator,'LineWidth',2);
181         hold on
182         p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
183             'K.','MarkerSize', value(2)+fator,'LineWidth',2);
184         p3 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
185             'K.','MarkerSize', value(1)+fator,'LineWidth',2);
186         [x y z] = sphere(50);
187         h = surfl(x, y, z);
188         set(h, 'FaceAlpha', 0)
189         daspect([1 1 1])
190         axis off
191         hold off
192     elseif ( value(4) ≤tol )
193         figure(2)
194         p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
195             'K.','MarkerSize', value(1)+fator,'LineWidth',2);
196         hold on
197         p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
198             'K.','MarkerSize', value(2)+fator,'LineWidth',2);
199         p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
200             'K.','MarkerSize', value(3)+fator,'LineWidth',2);
201         [x y z] = sphere(50);
202         h = surfl(x, y, z);
203         set(h, 'FaceAlpha', 0)
204         daspect([1 1 1])
205         axis off
206         hold off
207     end
208     Pts_aprox
209     alfa_aprox
210 end
211 toc
```

```
1 %EXEMPLO 11
2 clear all
3 close all
4 clc
5
6 %Intensidade das cargas
7 alpha(1,1) = 5;
```

```
8  alpha(2,1) = 5;
9  alpha(3,1) = 5;
10
11 %Numero de pontos do grid
12 L_Grid = 2.4;
13
14 %Grid gerado com o ball_grid
15 tam = ball_grid_count ( L_Grid, 1.0, [0,0,0] );
16 bg = ball_grid ( L_Grid, 1.0, [0,0,0], tam );
17 pt1 = 10;
18 pt2 = 41;
19 pt3 = 85;
20
21 p(1,:) = bg(1:3,pt1);
22 p(2,:) = bg(1:3,pt2);
23 p(3,:) = bg(1:3,pt3);
24
25 %Localizacoes das cargas
26 xyzst(1,1) = p(1,1);
27 xyzst(1,2) = p(1,2);
28 xyzst(1,3) = p(1,3);
29
30 xyzst(2,1) = p(2,1);
31 xyzst(2,2) = p(2,2);
32 xyzst(2,3) = p(2,3);
33
34 xyzst(3,1) = p(3,1);
35 xyzst(3,2) = p(3,2);
36 xyzst(3,3) = p(3,3);
37
38 %Variavel auxiliar para o tamanho das cargas no grafico
39 fator = 35;
40
41 %ruido (em porcentagem)
42 noise = 0;
43 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
44     mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
45
46 %utilizado para plotar a esfera
47 [x y z] = sphere(50);
48
49 %Plot dos graficos
50 figure(1)
51 p1 = plot3(xyzst(1,1),xyzst(1,2),xyzst(1,3),'K.',...
52     'MarkerSize', alpha(1,1) + fator,'LineWidth',2);
53 hold on
54 p2 = plot3(xyzst(2,1),xyzst(2,2),xyzst(2,3),'K.',...
```

```
55      'MarkerSize', alpha(2,1) + fator ,'LineWidth',2);
56  p3 = plot3(xyzst(3,1),xyzst(3,2),xyzst(3,3),'K.',...
57      'MarkerSize', alpha(3,1)+ fator ,'LineWidth',2);
58
59  h = surfl(x, y, z);
60  set(h, 'FaceAlpha', 0)
61  daspect([1 1 1])
62  axis off
63  hold off
64
65  alfa_aprox1 = alfa_aprox
66  Pts_aprox1 = Pts_aprox
67
68  r1 = alfa_aprox(1,1)/2;
69  r2 = alfa_aprox(2,1)/2;
70  r3 = alfa_aprox(3,1)/2;
71
72  value(1) = pi*(r1);
73  value(2) = pi*(r2);
74  value(3) = pi*(r3);
75
76  figure(2)
77  p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
78      'K.','MarkerSize', value(1) + fator,'LineWidth',2);
79  hold on
80  p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
81      'K.','MarkerSize', value(2) + fator ,'LineWidth',2);
82  p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
83      'K.','MarkerSize', value(3)+ fator ,'LineWidth',2);
84
85  h = surfl(x, y, z);
86  set(h, 'FaceAlpha', 0)
87  daspect([1 1 1])
88  axis off
89  hold off
90
91  noise = 5;
92  [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
93      mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
94
95  alfa_aprox2 = alfa_aprox
96  Pts_aprox2 = Pts_aprox
97
98  r1 = alfa_aprox(1,1)/2;
99  r2 = alfa_aprox(2,1)/2;
100 r3 = alfa_aprox(3,1)/2;
101
```

```
value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(3)

p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
    'r.','MarkerSize', value(1) + fator,'LineWidth',2);
hold on
p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
    'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
    'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);

h = surfl(x, y, z);
set(h, 'FaceAlpha', 0.1)
daspect([1 1 1])
axis off
hold off

noise = 10;
[alfa_aprox, Pts_aprox, q_star, q_star_noise] =...
    mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);

alfa_aprox3 = alfa_aprox
Pts_aprox3 = Pts_aprox

r1 = alfa_aprox(1,1)/2;
r2 = alfa_aprox(2,1)/2;
r3 = alfa_aprox(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);
figure(4)

p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
    'r.','MarkerSize', value(1) + fator,'LineWidth',2);
hold on
p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
    'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
    'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);

h = surfl(x, y, z);
set(h, 'FaceAlpha', 0.1)
daspect([1 1 1])
axis off
```

```
149 hold off
150
151 noise = 20;
152 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
153     mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
154
155 alfa_aprox4 = alfa_aprox
156 Pts_aprox4 = Pts_aprox
157
158 r1 = alfa_aprox(1,1)/2;
159 r2 = alfa_aprox(2,1)/2;
160 r3 = alfa_aprox(3,1)/2;
161
162 value(1) = pi*(r1);
163 value(2) = pi*(r2);
164 value(3) = pi*(r3);
165 figure(5)
166
167 p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
168     'r.','MarkerSize', value(1) + fator,'LineWidth',2);
169 hold on
170 p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),..
171 'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
172 p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
173     'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);
174
175 h = surfl(x, y, z);
176 set(h, 'FaceAlpha', 0.1)
177 daspect([1 1 1])
178 axis off
179 hold off
180
181 noise = 40;
182 [alfa_aprox, Pts_aprox, q_star, q_star_noise] =...
183     mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
184 alfa_aprox5 = alfa_aprox
185 Pts_aprox5 = Pts_aprox
186
187 r1 = alfa_aprox(1,1)/2;
188 r2 = alfa_aprox(2,1)/2;
189 r3 = alfa_aprox(3,1)/2;
190
191 value(1) = pi*(r1);
192 value(2) = pi*(r2);
193 value(3) = pi*(r3);
194
195 figure(6)
```

```
196 p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
197     'r.','MarkerSize', value(1) + fator,'LineWidth',2);
198 hold on
199 p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
200     'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
201 p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
202     'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);
203
204 h = surfl(x, y, z);
205 set(h, 'FaceAlpha', 0.1)
206 daspect([1 1 1])
207 axis off
208 hold off
209
210 toc
```

```
1  %EXEMPLO 13
2  clear all
3  close all
4  clc
5
6  tic
7  %Intensidade das cargas
8  alpha(1,1) = 9;
9  alpha(2,1) = 9;
10 alpha(3,1) = 9;
11
12 %Numero de pontos do grid
13 L_Grid = 2.4;
14
15 %Grid gerado com o ball_grid
16 tam = ball_grid_count ( L_Grid, 1.0, [0,0,0] );
17 bg = ball_grid ( L_Grid, 1.0, [0,0,0], tam );
18 pt1 = 1;
19 pt2 = 48;
20 pt3 = 80;
21
22 p(1,:) = bg(1:3,pt1);
23 p(2,:) = bg(1:3,pt2);
24 p(3,:) = bg(1:3,pt3);
25
26 %Localizacoes das cargas
27 xyzst(1,1) = p(1,1);
28 xyzst(1,2) = p(1,2);
29 xyzst(1,3) = p(1,3);
30
```

```
xyzst(2,1) = p(2,1);
xyzst(2,2) = p(2,2);
xyzst(2,3) = p(2,3);

xyzst(3,1) = p(3,1);
xyzst(3,2) = p(3,2);
xyzst(3,3) = p(3,3);

%Variavel auxiliar
fator = 35;

%ruido (em porcentagem)
noise = 40;
[alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
    mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);

[x y z] = sphere(50);

%Plot dos graficos
figure(1)
p1 = plot3(xyzst(1,1),xyzst(1,2),xyzst(1,3),'K.',...
    'MarkerSize', alpha(1,1) + fator,'LineWidth',2);
hold on
p2 = plot3(xyzst(2,1),xyzst(2,2),xyzst(2,3),'K.',...
    'MarkerSize', alpha(2,1) + fator ,'LineWidth',2);
p3 = plot3(xyzst(3,1),xyzst(3,2),xyzst(3,3),'K.',...
    'MarkerSize', alpha(3,1)+ fator ,'LineWidth',2);

h = surfl(x, y, z);
set(h, 'FaceAlpha', 0)
daspect([1 1 1])
axis off
hold off

alfa_aprox1 = alfa_aprox
Pts_aprox1 = Pts_aprox

r1 = alfa_aprox(1,1)/2;
r2 = alfa_aprox(2,1)/2;
r3 = alfa_aprox(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);

figure(2)
p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
```

```
78      'K.','MarkerSize', value(1) + fator,'LineWidth',2);
79  hold on
80  p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
81      'K.','MarkerSize', value(2) + fator ,'LineWidth',2);
82  p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
83      'K.','MarkerSize', value(3)+ fator ,'LineWidth',2);
84
85  h = surfl(x, y, z);
86  set(h, 'FaceAlpha', 0)
87  daspect([1 1 1])
88  axis off
89  hold off
90  adasdasd
91
92  noise = 5;
93  [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
94      mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
95
96  alfa_aprox2 = alfa_aprox
97  Pts_aprox2 = Pts_aprox
98
99  r1 = alfa_aprox(1,1)/2;
100 r2 = alfa_aprox(2,1)/2;
101 r3 = alfa_aprox(3,1)/2;
102
103 value(1) = pi*(r1);
104 value(2) = pi*(r2);
105 value(3) = pi*(r3);
106 figure(3)
107
108 p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
109     'r.','MarkerSize', value(1) + fator,'LineWidth',2);
110 hold on
111 p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
112     'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
113 p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
114     'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);
115
116 h = surfl(x, y, z);
117 set(h, 'FaceAlpha', 0.1)
118 daspect([1 1 1])
119 axis off
120 hold off
121
122 noise = 10;
123 [alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
124     mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
```

```
125
126 alfa_aprox3 = alfa_aprox
127 Pts_aprox3 = Pts_aprox
128
129 r1 = alfa_aprox(1,1)/2;
130 r2 = alfa_aprox(2,1)/2;
131 r3 = alfa_aprox(3,1)/2;
132
133 value(1) = pi*(r1);
134 value(2) = pi*(r2);
135 value(3) = pi*(r3);
136 figure(4)
137
138 p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
139     'r.','MarkerSize', value(1) + fator,'LineWidth',2);
140 hold on
141 p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
142     'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
143 p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
144     'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);
145
146 h = surfl(x, y, z);
147 set(h, 'FaceAlpha', 0.1)
148 daspect([1 1 1])
149 axis off
150 hold off
151
152 noise = 20;
153 [alfa_aprox, Pts_aprox, q_star, q_star_noise] =...
154     mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
155
156 alfa_aprox4 = alfa_aprox
157 Pts_aprox4 = Pts_aprox
158
159 r1 = alfa_aprox(1,1)/2;
160 r2 = alfa_aprox(2,1)/2;
161 r3 = alfa_aprox(3,1)/2;
162
163 value(1) = pi*(r1);
164 value(2) = pi*(r2);
165 value(3) = pi*(r3);
166
167 figure(5)
168 p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
169     'r.','MarkerSize', value(1) + fator,'LineWidth',2);
170 hold on
171 p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
```

```
    'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
    'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);

h = surfl(x, y, z);
set(h, 'FaceAlpha', 0.1)
daspect([1 1 1])
axis off
hold off

noise = 40;
[alfa_aprox, Pts_aprox, q_star, q_star_noise] = ...
    mainHelm_circular_3D(alpha, xyzst, noise, L_Grid);
alfa_aprox5 = alfa_aprox
Pts_aprox5 = Pts_aprox

r1 = alfa_aprox(1,1)/2;
r2 = alfa_aprox(2,1)/2;
r3 = alfa_aprox(3,1)/2;

value(1) = pi*(r1);
value(2) = pi*(r2);
value(3) = pi*(r3);

figure(6)
p1 = plot3(Pts_aprox(1,1),Pts_aprox(1,2),Pts_aprox(1,3),...
    'r.','MarkerSize', value(1) + fator,'LineWidth',2);
hold on
p2 = plot3(Pts_aprox(2,1),Pts_aprox(2,2),Pts_aprox(2,3),...
    'r.','MarkerSize', value(2) + fator ,'LineWidth',2);
p3 = plot3(Pts_aprox(3,1),Pts_aprox(3,2),Pts_aprox(3,3),...
    'r.','MarkerSize', value(3)+ fator ,'LineWidth',2);

h = surfl(x, y, z);
set(h, 'FaceAlpha', 0.1)
daspect([1 1 1])
axis off
hold off
toc
```